**VŠB – Technická univerzita Ostrava**
**Fakulta elektrotechniky a informatiky**
**Katedra informatiky**

# Automatizované a výkonnostní testování podnikových JavaEE systémů

# Automated and Performance Testing of Enterprise JavaEE Systems

**2015** **Bc. Pavel Rath**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání diplomové práce

Student: **Bc. Pavel Rath**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Automatizované a výkonnostní testování podnikových JavaEE systémů
Automated and Performance Testing of Enterprise JavaEE Systems

Zásady pro vypracování:

Cílem práce je vytvořit metodické pokyny, příklady a experimenty usnadňující vytváření testovacích scénářů a skriptů pro platformu JavaEE především se zaměřením na informační systém Edison.

Práce bude obsahovat:
1. Přehled nástrojů pro výkonnostní a automatizované testování.
2. Metodické pokyny pro vytváření automatizovaných a výkonnostních testů primárně určených pro systém Edison.
3. Popis infrastruktury systému Edison.
4. Přehled možností monitorování a sběru dat v systému Edison.
5. Příprava scénářů a skriptů pro automatizované a výkonnostní testování ve vybraných nástrojích.
6. Zhodnocení průběhu vytvořených testů.

Seznam doporučené odborné literatury:

[1] Rakitin, S. R. (2001), Software Verification and Validation for Practitioners and Managers, Second Edition, Artech House, Inc., Norwood, MA, USA.
[2] Fink, M. (2013), The Hitchhiker's Guide to Test Automation: Software Quality Assurance and Test Automation in Practice, CreateSpace Independent Publishing Platform.
[3] Beizer, B. (1995), Black-box Testing: Techniques for Functional Testing of Software and Systems, John Wiley & Sons, Inc., New York, NY, USA.
Dále dle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

## Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 29. 4. 2015

Pavel .....
podpis studenta

# Poděkování

Velice rád bych poděkoval Ing. Davidu Ježkovi, Ph.D. za odbornou pomoc, ochotu při konzultacích a věcné rady, které mi pomohly při vytváření této diplomové práce. Dále bych chtěl také poděkovat Ing. Radku Liebzeitovi a Ing. Petru Abrahamczikovi za informace a rady ohledně informačního systému Edison a také za součinnost během testovacího procesu. Poděkování patří také Ing. Martinu Lasoňovi, vedoucímu oddělení CIT-IS, za umožnění práce nad informačním systémem Edison. Na závěr bych chtěl také nesmírně poděkovat své přítelkyni a rodině za jejich podporu a pochopení.

# Abstrakt

Tato diplomová práce se zabývá problematikou automatizovaného funkčního a výkonnostního softwarového testování. Práce je zaměřena na testování webových aplikací napsaných v programovacím jazyce Java. Konkrétní příklady a výstupy diplomové práce jsou provedeny nad informačním systémem Edison. V práci jsou uvedeny jednotlivé pojmy testování a definice těchto pojmů. Práce dále obsahuje popis vybraných testovacích nástrojů spolu s postupem vytváření testů v těchto nástrojích. Je zde uveden také popis infrastruktury systému Edison a možnosti monitorování a sběru dat nad tímto systémem. Hlavním výsledkem práce je metodika testování určená pro systém Edison. Součástí práce jsou také ukázky automatizovaných funkčních a výkonnostních testů, spolu s popisem praktického testování a řešení konkrétního problému nad systémem Edison.



# Klíčová slova

Softwarové testování, metodika testování, testovací nástroje, Edison, monitorování, Java.

# Abstract

This master thesis is about automated functional and performance software testing. Thesis is focused on testing web applications which are written in Java programming language. Particular examples and outputs of this master thesis are done on information system Edison. There are stated individual terms of testing and also definition of these terms in this master thesis. Thesis also includes description of selected testing tools with described procedure of creation tests in these tools. There is also description of infrastructure of system Edison and possibilities of monitoring and collecting data from this system. Main result of this thesis is methodics of testing for information system Edison. This thesis also includes examples of automated functional and performance testing with description of practical testing and solving concrete problem with system Edison.



# Key words

Software testing, methodics of testing, testing tools, Edison, monitoring, Java.

# Seznam použitých zkratek

| Zkratka | Anglický význam | Český popis |
|---|---|---|
| **AJAX** | Asynchronous JavaScript and XML | Metoda asynchronního zpracování |
| **API** | Application Programming Interface | Aplikační programové rozhraní |
| **CSV** | Comma-separated values | Data oddělená čárkami |
| **FTP** | File Transfer Protocol | Protokol pro přenos souborů |
| **HTML** | HyperText Markup Language | Značkovací jazyk pro hypertext |
| **HTTP** | Hypertext Transfer Protocol | Protokol pro výměnu HTML dokumentů |
| **HTTPS** | Hypertext Transfer Protocol Secure | Zabezpečená verze protokolu HTTP |
| **IMAP** | Internet Message Access Protocol | Protokol pro přístup k emailové schránce |
| **IP** | Internet Protocol | Internetový protokol |
| **JDBC** | Java Database Connectivity | API Javy definující přístup k databázi |
| **JMX** | Java Management Extensions | Java technologie pro podporu monitorování aplikace |
| **JRE** | Java Runtime Environment | Celek složený z JVM a API |
| **JSON** | JavaScript Object Notation | Datový formát pro přenos dat |
| **JVM** | Java Virtual Machine | Virtuální stroj Javy |
| **LDAP** | Lightweight Directory Access Protocol | Protokol pro přístup k datům na adresářovém serveru |
| **OS** | Operating System | Operační systém |
| **POP3** | Post Office Protocol | Protokol pro stahování emailových zpráv |
| **SMTP** | Simple Mail Transfer Protocol | Protokol pro přenos emailových zpráv |
| **SNMP** | Simple Network Management Protocol | Protokol pro správu a sběr dat |
| **SOA** | Service Oriented Architecture | Architektura orientovaná na služby |
| **SOAP** | Simple Object Access Protocol | Protokol pro výměnu XML dat |
| **SQL** | Structured Query Language | Jazyk pro práci s databázemi |
| **SSO** | Single Sign On | Systém jednotného přihlášení |
| **URL** | Uniform Resource Locator | Identifikátor umístění zdrojů |
| **WAS** | WebSphere Application Server | Aplikační server WebSphere |
| **WPS** | WebSphere Portal Server | Portálový server WebSphere |

# Obsah

# 1 Úvod

Testování softwaru je dnes již nedílnou součástí každého vývojového cyklu a procesu. K tomuto stavu vedla dlouhá a náročná cesta, která začala v sedmdesátých a osmdesátých letech minulého století, kdy náklady na údržbu softwaru začaly převyšovat náklady na tvorbu softwaru. Většina lidí si pod pojmem testování softwaru představí druhořadou práci, která je pouze o tom, že tester sedí u počítače a rutinně provádí manuální testování aplikace. Ve skutečnosti se ovšem jedná o profesionální práci, která má své nenahraditelné místo při vývoji softwaru. Tato práce zahrnuje několik činností jako je analýza, návrh, programování testovacích scénářů, jejich následné vyhodnocení, práce s testovacími nástroji, automatizace testů a další.

Tato diplomová práce je zaměřena na testování informačního systému Edison. Jedná se o celouniverzitní informační systém pro podporu výuky na Vysoké škole báňské – Technické univerzitě Ostrava. Název tohoto systému vznikl jako akronym z anglického „Education Information System on Net“. Volně tedy přeloženo jako webový informační systém pro vzdělávání.

Systém Edison byl kompletně vyvinut pracovníky oddělení Centra informačních technologií (CIT) na VŠB – TU Ostrava. Do provozu byl systém nasazen v roce 2008. Od té doby prošel systém několika změnami a byl rozšířen do mnohonásobně větších rozměrů. Tento systém kompletně nahradil studijní indexy, jeho pomocí jsou evidovány veškeré studijní výsledky a informace o studentovi. Jeho prostřednictvím probíhá volba osobního studijního plánu, volba rozvrhů, volba a odevzdání závěrečné práce a spousta dalších akcí a procesů.

Cílem této diplomové práce je prozkoumat možnosti testování nad systémem Edison a také samotný systém Edison z jeho technické stránky. Dále je pak cílem vytvořit postupy, příklady a metodické pokyny pro testování, které by usnadnily vytváření a spouštění testů pro další účely.

Na začátku této práce budou vymezeny některé základní pojmy testování, následovat bude popis vybraných testovacích nástrojů, a to jak pro automatizované funkční, tak i pro výkonnostní testování. Výběr nástrojů bude zahrnovat komerční i open-source nástroje. Současně bude u každého nástroje vytvořen i návod pro tvorbu testů. V další kapitole bude popsána architektura systému Edison a také infrastruktura, na které systém běží. Dále bude práce obsahovat popis monitorovacích nástrojů, které jsou nad systémem nasazeny a popsány další možnosti sběru dat. V rámci práce vzniknou také metodické pokyny pro návrh, tvorbu, spouštění a vyhodnocování automatizovaných funkčních a výkonnostních testů. Na závěr budou v práci uvedeny příklady ukázkových testů, které mohou sloužit jako referenční testy pro tvorbu budoucích testů. Současně bude také popsán průběh a zkušenosti, případně problémy s praktickým testováním systému Edison.

# 2 Základní pojmy testování

## 2.1 Testování softwaru

Testování je aktivita vykonávaná za účelem vyhodnocení kvality produktu a k jejímu zlepšení na základě identifikování vad a problémů. Tato aktivita spočívá v dynamickém ověřování chování programu v konečném počtu testovacích případů. Tyto testovací případy jsou vybírány z množiny velkého počtu možností a kombinací vstupů nad danou funkční množinou. Dynamickým ověřováním se myslí ověřování skutečné situace podle aktuálních vstupů. [4]

## 2.2 Manuální testování

Manuální testování je proces, kde tester vystupuje v roli koncového uživatele a ověřuje správnost chování daného systému. Během testování se zpravidla postupuje podle předem vytvořeného testovací případu, který popisuje jednotlivé kroky, dané vstupy a očekávané výsledky. Veškeré odlišnosti oproti tomuto testovacímu případu musí zaznamenávat a vyhodnocovat samotný tester. Tyto testy jsou vhodné pouze pro malé projekty a aplikace anebo pro tzv. exploratorní testování.

## 2.3 Automatizované testování

„Za automatizované testování softwaru se považuje takové, kdy je část testovacího procesu, nebo i celý tento proces, prováděn bez přímého působení člověka a to pomocí specializovaného softwaru. Existuje celá řada programů, jejichž úkolem je samostatně provádět některou z činností, které jsou součástí testování. Nejznámější jsou programy, které dokáží nahradit fyzické klikání testerů do testované aplikace. Ale existují i programy, které automatizují například instalaci a konfiguraci testované aplikace, další jsou schopné porovnávat výsledky opakovaných testů mezi sebou a hlásit případné rozdíly, existují i nástroje pro generování testovacích případů z analýzy atd.“ [15]

Automatizované testování se používá u středních až velkých projektů, kde již není možné si vystačit s pouhým manuálním testováním, protože je potřeba spouštět velké množství testů a tyto testy také opakovat. Automatizované testování se také využívá u výkonnostních testů, protože u nich je možnost vytvoření větší zátěže na aplikaci manuálně prakticky nemožná.

## 2.4 Exploratorní testování

Exporatorní testování bývá někdy také označováno jako ad-hoc testování a jedná se o proces testování, během kterého se současně poznává a učí testovaná aplikace, navrhuje a vykonává se test. Nejedná se tedy o proces, který by přesně sledoval definované kroky, ale lze jej použít právě pro přesné definování těchto kroků pro další testování. [14]

## 2.5 Funkční testování

Funkční testování je proces, který má za primární úkol ověřit, že část programu poskytuje stejný výstup, jaký je požadován koncovými uživateli nebo zákazníky. Typicky se jedná o vyhodnocování a porovnávání funkcí programu s dodanými požadavky. Funkční testování také většinou ověřuje použitelnost systému. [16]

## 2.6 Systémové testování

Systémové testování bývá většinou označováno také jako nefunkční testování. „Nefunkční testy spočívají v testování všech vlastností aplikace, které přímo nesouvisí s jejími funkcemi, ale zároveň jsou podstatné pro její správné fungování. Řadí se sem především výkonové testování.“ [17]

Dalšími testy, které patří do této kategorie, jsou např. testy bezpečnostní, testy úložiště, testy instalovatelnosti, testy zotavení a další.

### 2.6.1 Výkonnostní testování

Pojmy vztahující se k výkonnostním testům nejsou zcela jednotné a mírně se liší v každé publikaci. Existuje několik typů testů, kterými se ověřuje výkon dané aplikace. Souhrnně se tyto testy většinou skrývají pod názvem výkonnostní testování (v angličtině Performance Testing). [1]

Na výkonnostní testování se dá použít obecná definice: „Je to testování za účelem vyhodnocení míry času zpracování a propustnosti u systému nebo komponenty, která plní své funkce v daných omezujících hranicích.“ [1]

#### 2.6.1.1 Zátěžové testování

Zátěžové testování v angličtině označované jako Load Testing, je typ výkonnostního testování, jehož účelem je vyhodnocení chování komponent při zvyšující se zátěži. Cílem většinou bývá určení hranice, kterou je systém nebo komponenta ještě schopen zvládnout. Typicky se vytváří pomocí kombinace více reálných akcí, které jsou prováděny více paralelními uživateli najednou. U těchto testů se měří čas odezvy nebo propustnost. [1]

#### 2.6.1.2 Stresové testování

V angličtině je stresové testování označováno jako Stress Testing a jedná se o typ výkonnostního testování, jehož účelem je vyhodnotit chování systému při zátěži na hranici nebo za hranicí systému. Cílem je ujistit se, že doby odezvy, spolehlivost a funkcionalita se budou snižovat postupně a očekávaným způsobem např. za pomoci hlášek pro uživatele místo toho, aby došlo k pádu systému a ztrátě nebo poškození dat. [1]

U těchto testů je obzvláště důležité spouštět je ve zvláštním kontrolovaném prostředí tak, aby bylo možné testovat nejrůznější situace a zároveň nedošlo k ohrožení dalších systémů.

#### 2.6.1.3 Škálovatelné testování

Škálovatelné testování, které je v angličtině označováno výrazem Scalability Testing, navazuje na stresové testování. Účelem je nalézt úzká hrdla systému a po navýšení systémových prostředků ověřit, zda byl problém odstraněn či nikoliv. [1]

#### 2.6.1.4 Vytrvalostní testování

Vytrvalostní testování, v angličtině označované jako Endurance Testing, vytváří na systém několikanásobně vyšší zátěž, než je normální provozní zátěž, a to po dlouhou dobu. Účelem je ověřit, že se systém nezhroutí ani po velkém počtu transakcí, které probíhají delší dobu. Tento test umí odhalit např. chyby při alokaci paměti a jiných zdrojů. [1]

#### 2.6.1.5 Testování špiček

Testování špiček, v angličtině známé pod názvem Spike Testing, slouží k ověření stability systému při nečekaně velkém nárůstu akcí nebo uživatelů. Příkladem může být situace, kdy systém má plánovaný výpadek a uživatelé k němu ztratí přístup. Po ukončení výpadku se všichni uživatelé znovu najednou připojí. Cílem tohoto testu je také ověřit, že se systém z tohoto náporu vzpamatuje a vrátí se do normální funkčnosti. [1]

#### 2.6.1.6 Testování spolehlivosti

Účelem testování spolehlivosti, jehož anglický název je Reliability Testing, je prověřit, zda systém unese danou zátěž, která by měla odpovídat reálné zátěži na systém, po určitou delší dobu. [1]

## 2.7 Regresní testování

Regresním testováním je nazýván proces znovu spouštění testů systému nebo komponent, které již dříve úspěšně proběhly v době, kdy byly v aplikaci provedeny změny, a je potřebné zjistit, zda dříve ověřená funkcionalita je stále bez vady. [4]

Důležitým faktorem je zde správný výběr testů pro regresní testování, protože množina dříve úspěšných testů může být na tolik velká, že při změně systému je nereálné znovu vykonat všechny testy.

## 2.8 Testovací nástroje

Neodmyslitelnou součástí testování softwaru jsou testovací nástroje, které pomáhají s vytvářením, spouštěním a vyhodnocováním testů. Díky testovacím nástrojům je možné vytvářet automatizované testy, které zefektivní samotné testování, ušetří čas běhu testů a tím šetří i vynaložené finance. Existuje velké množství testovacích nástrojů zaměřených na jednotlivé typy testování, a to jak v placených verzích, tak i v open-source řešeních.

# 3 Nástroje pro výkonnostní a automatizované funkční testování

## 3.1 Apache JMeter

### 3.1.1 Popis

JMeter je desktopová open-source aplikace, která je napsaná v programovacím jazyce Java. To zajišťuje její přenositelnost a funkčnost na různých platformách. Hlavním účelem aplikace je výkonnostní testování. Původně byla aplikace určena pouze pro testování webových aplikací bez ohledu na to, v jakém jazyce jsou napsány. Nyní aplikace JMeter umožňuje testovat několik různých aplikací a aplikačních protokolů, mezi které patří např. webové aplikace s využitím HTTP i HTTPS, SOAP, FTP, databáze přes JDBC, LDAP, emailové aplikace s využitím protokolů SMTP, POP3, IMAP a další. [5]

Aplikace dokáže pro testování webových aplikací simulovat, ale ne nahrazovat, webový prohlížeč. Aplikace umí simulovat cache prohlížeče, pracovat s cookie soubory a u zaznamenaných odpovědí stránek umí interpretovat HTML kód. Aplikace naopak neumí spouštět JavaScriptový kód nalezený v HTML stránkách.

První verze JMeter 1.0.2 byla vydána na začátku roku 2001. Aktuální verze, ve které jsem tvořil testy, je verze 2.12 z podzimu roku 2014. Aktuální verze potřebuje pro svou funkčnost Javu minimálně ve verzi 6. [6]

### 3.1.2 Princip a práce s nástrojem

Aplikace JMeter se ovládá přes grafické rozhraní. Každý test se skládá ze dvou základních částí. Tou první je *Test Plan*, kde jsou uvedeny všechny části testu, který je poté spouštěn. Druhou částí je *Workbench*, který obsahuje netestové části.

Hlavním prvkem testu je *Thread Group*, který obsahuje jednotlivé požadavky a umožnuje nastavit, pro kolik uživatelů (kolik vláken) má být daný test spuštěn. Lze také nastavit, kolikrát má být daná část testu spuštěna za sebou, nebo lze naplánovat spouštění pomocí prvku *Scheduler*.

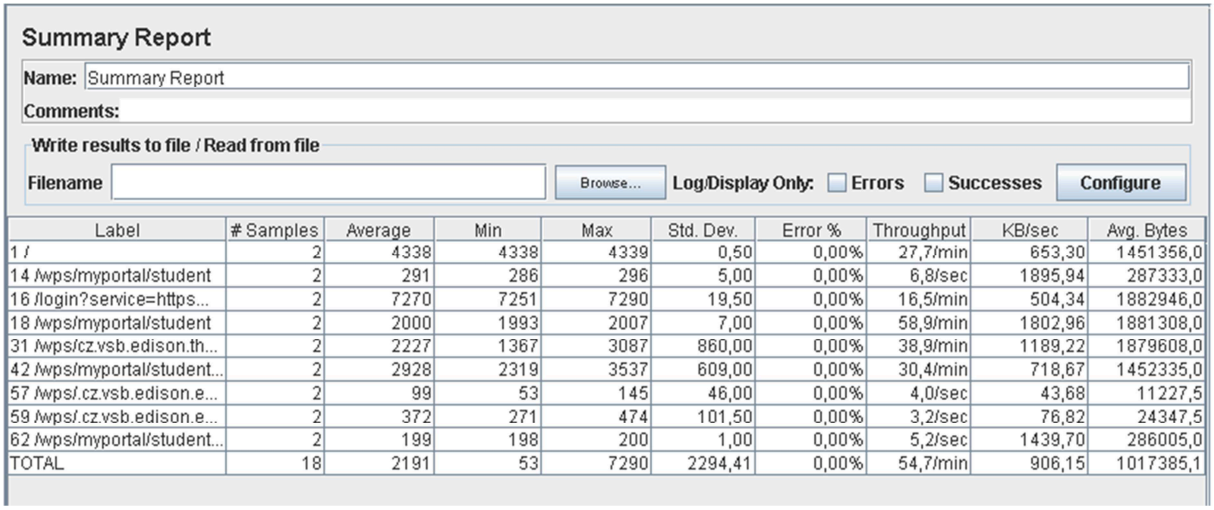
Samotný *Thread Group* obsahuje jednotlivé HTTP požadavky na webovou aplikaci, které jsou postupně volány. U jednotlivých požadavků je potřeba nastavit cestu, protokol, metodu a případné parametry. Je možné také nastavit spoustu dalších parametrů, jako jsou např. kódování, jednotlivé timeouty, jestli se mají sledovat jednotlivá přesměrování atd. Další prvkem, který je vhodné použít v daném *Thread Group* je *HTTP Request Defaults*, který umožňuje nastavit výchozí hodnoty pro všechny následující požadavky – např. jméno serveru a port.

Pokud se má testovat aplikace, která pro svou práci vyžaduje cookie soubory, tak je potřeba přidat prvek *HTTP Cookie Manager*. Tento prvek umožňuje nastavit *Cookie Policy* a *Implementaci*.

Pro spuštění základního testu stačí pouze výše uvedené položky. Pro sledování výsledku testů se používají prvky z kategorie *Listener*. Prvek *View Results Tree* umožňuje sledovat jednotlivé požadavky, které byly volány na aplikaci a také jejich odpovědi. Prvek je vhodný hlavně pro fázi ladění testů, kdy je možné sledovat, jaké požadavky jsou ve skutečnosti volány a jestli přicházejí

očekávané odpovědi. Především pro vyhodnocování testů je vhodný prvek *Summary Report*, který změří u jednotlivých požadavků délku jejich odpovědi a další statistické údaje jako průměr, směrodatnou odchylku, propustnost atd. Tyto údaje je možné vyexportovat do CSV souboru a následně z něj vytvořit požadované grafy.

Pro vytváření pokročilejších testů je nutné použití dalších prvků, které umožňují dynamicky měnit obsah požadavků, kontrolovat správnost odpovědi nebo třeba spouštět některé požadavky ve smyčce. Pro dynamický obsah požadavků (ať už cesty nebo parametrů) se využívá proměnných, které se v programu používají ve formě `${název_proměnné}`. Tyto proměnné lze nastavit buď globálně pro celý test, nebo je plnit z CSV souboru anebo pomocí regulárních výrazů extrahovat některou část předchozí odpovědi. Správnost odpovědi se kontroluje pomocí prvků z kategorie *Assertions*. Těmito prvky lze kontrolovat např. kód odpovědi (Response Code), velikost odpovědi, textový obsah odpovědi a další.




| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| 1 / | 2 | 4338 | 4338 | 4339 | 0,50 | 0,00% | 27,7/min | 653,30 | 1451356,0 |
| 14 /wps/myportal/student | 2 | 291 | 286 | 296 | 5,00 | 0,00% | 6,8/sec | 1895,94 | 287333,0 |
| 16 /login?service=https... | 2 | 7270 | 7251 | 7290 | 19,50 | 0,00% | 16,5/min | 504,34 | 1882946,0 |
| 18 /wps/myportal/student | 2 | 2000 | 1993 | 2007 | 7,00 | 0,00% | 58,9/min | 1802,96 | 1881308,0 |
| 31 /wps/cz.vsb.edison.th... | 2 | 2227 | 1367 | 3087 | 860,00 | 0,00% | 38,9/min | 1189,22 | 1879608,0 |
| 42 /wps/myportal/student... | 2 | 2928 | 2319 | 3537 | 609,00 | 0,00% | 30,4/min | 718,67 | 1452335,0 |
| 57 /wps/.cz.vsb.edison.e... | 2 | 99 | 53 | 145 | 46,00 | 0,00% | 4,0/sec | 43,68 | 11227,5 |
| 59 /wps/.cz.vsb.edison.e... | 2 | 372 | 271 | 474 | 101,50 | 0,00% | 3,2/sec | 76,82 | 24347,5 |
| 62 /wps/myportal/student... | 2 | 199 | 198 | 200 | 1,00 | 0,00% | 5,2/sec | 1439,70 | 286005,0 |
| TOTAL | 18 | 2191 | 53 | 7290 | 2294,41 | 0,00% | 54,7/min | 906,15 | 1017385,1 |

Obrázek 3.1: *Ukázka výstupu z programu JMeter*

### 3.1.3 Postup vytváření testů

Nejjednodušším a zřejmě i nejlepším způsobem, jak vytvořit test, je použít *HTTP Test Script Recorder*, který se přidává v části *Workbench*. Tento *Recorder* vytvoří lokální proxy server, který umí zachytávat jednotlivé požadavky při procházení webu. Výhodou je, že se nemusí přidávat jednotlivé požadavky ručně a hlídat u nich všechny parametry, které se jim musí předat.

V nastavení záznamníku *Recorder* se nastaví port, na kterém má proxy server běžet a pomocí regulárního výrazu se nastaví URL, které se nemají zaznamenávat – slouží k tomu, aby se zbytečně nenahrávaly požadavky na obrázky, kaskádové styly nebo soubory s JavaScripty. JMeter má předdefinovaný filtr doporučených přípon, které se nemají zachytávat. Patří zde: bmp, css, js, gif, ico, jpg, png, swf a woff soubory.

V části *Test Plan* se přidá prvek *Thread Group* a do něj *HTTP Request Defaults*, kde se nastaví jméno serveru, který je testován a na který se posílají požadavky. Spustí se *Recorder*, otevře se prohlížeč a nastaví se používání proxy serveru na adrese *localhost* a portu, který byl zvolen v nastavení JMeteru. V prohlížeči se postupně projdou jednotlivé stránky, které se mají testovat.

Pokud se přistupuje na stránky přes protokol HTTPS, je potřeba pro daný prohlížeč schválit certifikát JMeteru, který proxy server používá.

Po skončení nahrávání se musí *Recorder* v JMeteru zastavit. V části *Thread Group* přibyly jednotlivé nahrané požadavky. Nyní je potřeba požadavky zkontrolovat, nežádoucí promazat, upravit části požadavků, které mají být dynamicky doplňovány a případně k jednotlivým požadavkům přidat prvky pro ověření, že odpověď serveru je taková, jaká se očekávala. Pokud se testuje webová aplikace, která využívá cookie souborů, je potřeba ještě přidat a nastavit prvek *HTTP Cookie Manager*. Pro testování webové aplikace Edison jsem použil politiku dle *rfc2109* a implementaci *HC3CookieHandler*. Jestliže se při testování přistupuje na stránky po přihlášení, je zpravidla nutné pro správné přihlášení dynamicky nastavit parametr s bezpečnostním tokenem, který se automaticky generuje, jinak by při přehrávání testu přihlášení neproběhlo úspěšně.

Poslední částí, kterou je potřeba přidat je monitorování výsledků pomocí prvků *View Results Tree* a *Summary Report*.

### 3.1.4 Výhody a nevýhody

+ open-source nástroj
+ multiplatformní
+ velká množina aplikací, které lze testovat
+ omezení zaznamenávaných požadavků podle typu
+ dobrá podpora pro dynamické úpravy
+ velký počet tutoriálů a návodů

− při nahrávání se automaticky nehlídají dynamické části požadavků
− horší ladění testů
− naměřené výsledky v testech nezůstávají uloženy

## 3.2 Eclipse TPTP

### 3.2.1 Popis

Aplikace s celým názvem *Eclipse Test and Performance Tools Platform* je open-source nástroj sloužící pro vytváření testů a výkonnostní testování. Platforma TPTP existuje jako rozšiřující plugin do vývojového prostředí Eclipse. Existuje také přímo verze Eclipse TPTP, která má tento plugin předinstalován. Aplikace je napsána v programovacím jazyce Java, takže je opět možné spouštět nástroj na různých platformách a operačních systémech. [7]

První verze Eclipse TPTP 1.0.3 byla vydána na podzim roku 2003. Poslední dostupná verze je 4.7.2 a byla vydána v únoru roku 2011. Jedná se o verzi, která je integrovaná do verze Eclipse Helios SR2. Po vydání této verze bylo rozhodnuto, že projekt TPTP bude archivován a nebude již dále vyvíjen.

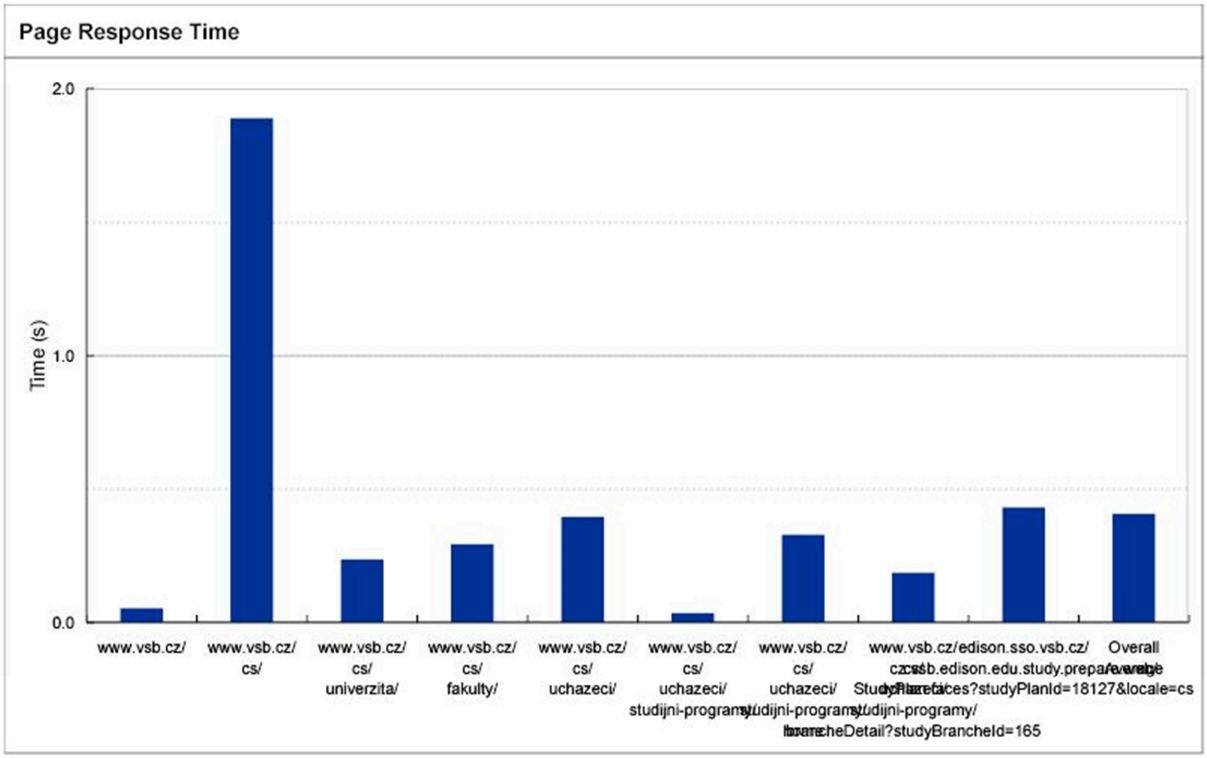
Nutno podotknout, že pro správnou funkci programu je potřeba mít nainstalovanou Javu nejvýše ve verzi 6. Pro zobrazování grafů je dále třeba mít nainstalovaný program Adobe SVG Viewer.

### 3.2.2 Princip a práce s nástrojem

S nástrojem se pracuje přes vývojové prostředí Eclipse, kde po nainstalování pluginu je k dispozici perspektiva *Test*. Pro výkonnostní testování webových aplikací se používá projekt typu *URL Test*. Podobně jako u programu JMeter existují dvě možnosti, jak test tvořit. První možností je manuálně přidat jednotlivé požadavky a vyplnit jejich parametry. Druhou možností je použít zabudovaný záznamník *Recorder*, který v principu funguje stejně jako u JMeteru – vytvoří se lokální proxy server, který zachycuje veškeré požadavky a uloží je do programu. Při nahrávání je nevýhodou, že není možné omezit požadavky, které se mají nahrávat, takže je následně nutné nahrané požadavky manuálně zkontrolovat a promazat.

Před spuštěním testu je třeba ze seznamu požadavků vygenerovat zdrojový kód, který využívá JUnit framework. Pokud je potřebné dynamicky upravovat některé části požadavků, spouštět některé požadavky v cyklu apod., musí se tyto funkčnosti doprogramovat do vygenerovaného zdrojového kódu, protože jejich nastavení přes grafické rozhraní není možné.

Po spuštění testu se automaticky vygeneruje zpráva s přehledem jednotlivých událostí a s výsledkem, zda test proběhl úspěšně či nikoliv. Zpráva také obsahuje časy odezvy pro jednotlivé požadavky. Dále je možné vygenerovat tzv. *Report*, což je pouze graf s danými údaji – např. doba odezvy jednotlivých požadavků.




Obrázek 3.2: *Ukázka výstupu z programu Eclipse TPTP*

### 3.2.3 Postup vytváření testů

Před samotným vytvořením testu pomocí záznamníku *Recorder* je potřeba nastavit, který prohlížeč se má použít a na jakém portu má proxy server běžet. Nyní se v prostředí Eclipse TPTP vytvoří nový Java projekt a v něm se vytvoří nový *Test Element* typu *TPTP Test From Recording*.

Následně se automaticky spustí nahrávání. Pokud byl v nastavení vybrán konkrétní prohlížeč, automaticky se otevře a nastaví tak, aby se používal proxy server na daném portu. Pokud nebylo použití prohlížeče nastaveno, musí se spustit a nastavit manuálně.

Nyní se projdou v prohlížeči dané webové stránky tak, aby se v aplikaci zachytily požadavky, které mají být obsažené v testu. Po ukončení nahrávání je třeba vymazat ze seznamu požadavků ty, které nemají v testu být. Nepříjemné je to, že požadavků se většinou zachytí opravdu hodně a rozlišovat je lze v podstatě jenom podle absolutní cesty. U jednotlivých požadavků je také možné upravit jednotlivé parametry. Nastavit lze ale pouze statické hodnoty.

Po úpravě seznamu požadavků se nechá automaticky vygenerovat kód testu. Pokud se mají v testu dělat nějaké pokročilejší změny, musí se upravit tento vygenerovaný kód.

### 3.2.4 Výhody a nevýhody

+ open-source nástroj
+ multiplatformní

− již neaktuální verze
− prakticky neexistují návody a tutoriály
− nelze zobrazit jednotlivé odpovědi
− nelze ladit testy
− nedostatečné zobrazování výsledků testování
− při nahrávání nelze omezit typy požadavků
− obtížná a nepřehledná úprava nahraných skriptů

## 3.3 IBM Rational Performance Tester

### 3.3.1 Popis

Jedná se o aplikaci z komerčního balíku nástrojů vyvíjených firmou IBM. Performance Tester je určený pro výkonnostní testování webových aplikací typu HTTP, SAP, Citrix, socket, SOA a další. Performance Tester je založen na vývojovém prostředí Eclipse a v nejnovější verzi je dostupný pro platformy Windows, Linux a Mac OS. [12]

První verze Performance Testeru byla označována jako 6.1 a byla vydána v květnu roku 2005. Aplikace v označování verzí navazuje na Rational Performance Tester, který ovšem nesouvisí se současným IBM nástrojem. Aktuální verze je 8.7 a vyšla v březnu 2015. [13]

Ke tvorbě svých testů jsem použil verzi 8.5.1.3.

### 3.3.2 Princip a práce s nástrojem

Podobně jako u ostatních nástrojů používá i Performance Tester pro vytvoření testu zabudovaný záznamník *Recorder*, který i zde funguje na principu vestavěného proxy serveru, který zachycuje jednotlivé požadavky. Výhodou je, že se nemusí nic konfigurovat a aplikace si sama automaticky nastaví daný proxy server i prohlížeč, ve kterém bude záznam probíhat. Doporučené prohlížeče jsou Internet Explorer a Mozilla Firefox.

Během nahrávání aplikace sama automaticky seskupuje požadavky do skupin, které pojmenuje podle dané stránky. Seskupení funguje tak, že při novém požadavku se vytvoří nová skupina a všechny další požadavky, které jsou zachyceny po určitou dobu, se přiřadí do této skupiny. Výsledkem pak je, že jsou požadavky ke každé stránce logicky seskupeny. Čas, po který se jednotlivé požadavky seskupují do jedné skupiny lze nastavit. Ve výchozím nastavení je tento čas 5 sekund.

Práce s nástrojem je ulehčena v tom, že aplikace automaticky rozpoznává a nahrazuje dynamické části požadavků. To znamená, že většinou po zaznamenání testů bude test spustitelný bez dalších úprav. Nástroj samozřejmě umožňuje nastavit, zda mají být dané části nahrazovány, případně přidat vlastní nahrazování. Nahrazovat lze za statickou hodnotu nebo je možné využít úložiště *Datapool*, což je v podstatě tabulka dat, která lze při testování využívat. Při použití úložiště *Datapool* je možné nastavit, jak se mají jednotlivá data vybírat, zda sekvenčně nebo náhodně a zda má být *Datapool* sdílený pro všechna vlákna (uživatele), pro která je test spuštěn, nebo každé vlákno má mít svou instanci úložiště *Datapool*. Další možností, jak upravovat jednotlivé části požadavků je použít vlastní kód napsaný v jazyce Java, který bude vracet hodnoty pro jednotlivé požadavky.

Pro ověření správnosti jednotlivých požadavků se využívá verifikačních bodů. Ověřovat lze obsah odpovědi pomocí regulárního výrazu, velikost odpovědi s definovanou tolerancí oproti velikosti zaznamenané při nahrávání, kód odpovědi a titulek stránky.

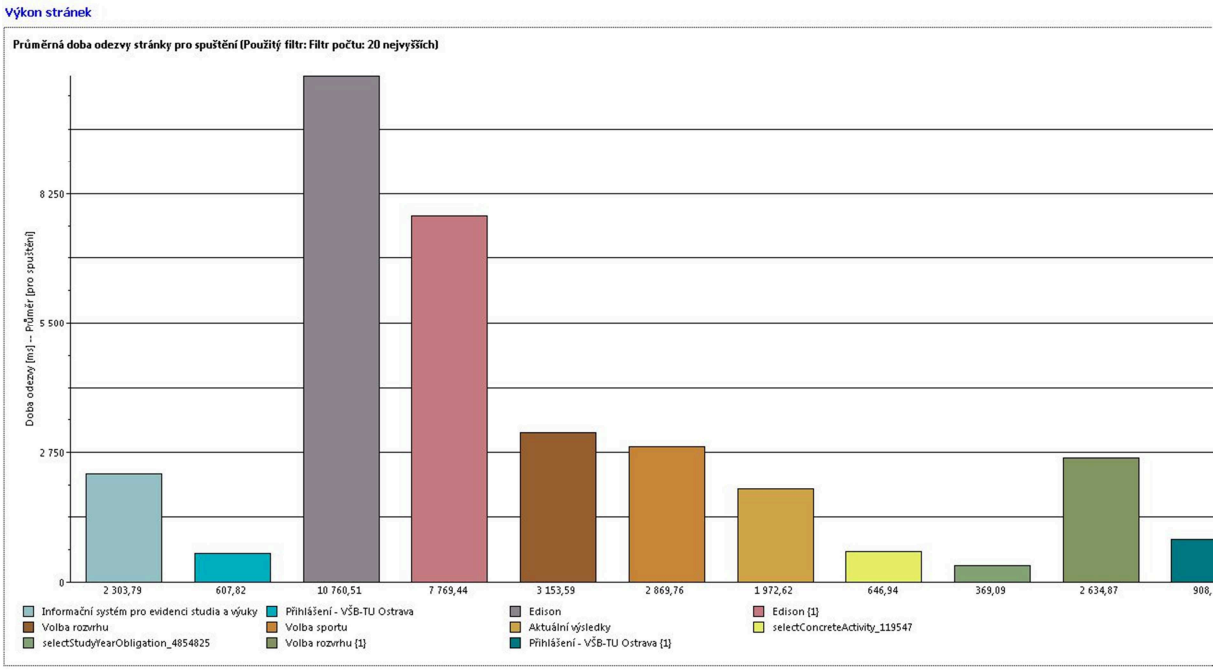
Průběh testů je možné ovlivňovat dalšími prvky, které lze vložit mezi jednotlivé požadavky. Může se vložit podmínka, cyklus nebo náhodný selektor. Je zde také možnost vkládat komentáře, což nám pomůže hlavně u komplexnějších testů.

Pro vytvoření samotné zátěže slouží tzv. *Schedule* neboli plán testu, ve kterém se spouští jednotlivé testovací skripty pro definované skupiny uživatelů. Skupiny uživatelů můžou být různě velké, avšak bez rozšiřující licence lze nastavit maximálně pět uživatelů.

Po spuštění testu se začne automaticky v reálném čase generovat report s dosaženými výsledky. Data jsou prezentována pomocí tabulek a grafů a tyto výstupy lze v aplikaci i upravit. Obrovskou výhodou Performance Testeru je, že je možné k monitorování přidat i další zdroje jako např. monitorování stroje, ze kterého testování spouštíme, monitorování serveru, na kterém aplikace běží atd. Obecně pro monitorování lze importovat zdroje z nástrojů:

- Apache HTTP Server Monitoring,
- Apache Tomcat Application Server Monitoring,
- IBM DB2 Monitoring,
- IBM Tivoli Monitoring,
- IBM WebSphere PMI Monitoring,
- JBoss Application Server Monitoring,
- JVM Monitoring,
- Oracle Database Monitoring,
- Oracle WebLogic Server Monitoring,
- SAP NetWeaver WebApplication Server Monitoring,
- SNMP Monitoring,
- UNIX rstatd monitoring,
- Windows Performance Monitor.

Generování zátěže na aplikaci je poměrně systémově náročná akce pro stanici, ze které se test spouští, a proto je možné tuto zátěž rozložit na více strojů. K tomu je potřeba na jednotlivé stroje doinstalovat aplikaci *Rational Performance Tester Agent Controller*, která se spouští jako služba. Po správném nastavení, se po spuštění testu Performance Tester připojí na definované stroje s touto službou a z těchto strojů se pak generuje zátěž na cílovou aplikaci.




Souhrn výkonu

| | Doba odezvy [ms] -- Minimum [pro spuštění] | Doba odezvy [ms] -- Průměr [pro spuštění] | Doba odezvy [ms] -- Směrodatná odchylka [pro spuštění] | Doba odezvy [ms] -- Maximum [pro spuštění] | Pokusy -- Četnost [za sekundu] [pro spuštění] | Pokusy -- Počet [pro sp |
|---|---|---|---|---|---|---|
| Informační systém pro evidenci studia a výuky | 466 | 2 303,8 | 1 356,6 | 4 496 | 3,22 | 393 |
| Přihlášení - VŠB-TU Ostrava | 90 | 607,8 | 439,9 | 1 473 | 3,21 | 392 |
| Edison | 3 133 | 10 760,5 | 4 445,5 | 19 352 | 3,19 | 389 |

Obrázek 3.3: *Ukázka výstupu z programu IBM RPT*

### 3.3.3 Postup vytváření testů

Pro vytvoření testu v aplikaci se přidá nový projekt typu *Projekt test výkonu*. Do projektu se přidá *Nový test ze záznamu*. Pro webové aplikace se vybere typ *Test HTTP* a prohlížeč, ve kterém bude probíhat nahrávání. Po jeho vytvoření se automaticky spustí vybraný prohlížeč a nastaví se v něm proxy server. V prohlížeči se zobrazí stránka s informacemi a doporučeními. Před začátkem testu je vhodné v prohlížeči promazat dočasné soubory proto, aby zachytávání požadavků pro test a jeho následné přehrání bylo reálné, tedy aby se při prvním požadavku načetly všechny zdroje a nenačítalo se nic z cache paměti prohlížeče. Naopak u dalších požadavků už se nebudou načítat všechny zdroje znovu, pokud již jsou uložené, přesně tak, jako v reálné situaci nového uživatele. Během nahrávání se postupně prokliká aplikace tak, aby se zachytily jednotlivé kroky testu. Je potřeba neklikat příliš rychle, aby se všechny požadavky správně zachytily a také, aby se dvě stránky nesjednotily do jedné skupiny. To znamená, že před přechodem na novou stránku musí uplynout aspoň pět sekund (pokud nebyl změněn čas v nastavení).

Pro ukončení nahrávání stačí zavřít prohlížeč. Nástroj nyní automaticky převede zachycené akce do testovacího skriptu. Automaticky také pohlídá dynamické části požadavků. Nyní se

zkontrolují jednotlivé požadavky, případně promažou nežádoucí požadavky. Na požadavky se nastaví také verifikační body podle našich potřeb. U požadavků se zkontrolují části, které se nahrazují automaticky, případně se nastaví další části, které se mají nahradit, a také hodnoty, kterými se mají nahradit. Hodnoty, kterými se nahrazuje, můžou být:

- staticky vyjmenované,
- uložené v proměnné testu,
- přiřazené z úložiště *Datapool*,
- definované pomocí vlastního kódu.

Pokud se má dynamicky vybírat část z předchozí odpovědi pro daný požadavek, tak je ve stromové struktuře požadavků potřebné vybrat odpověď daného požadavku. V pravé části se zobrazí obsah odpovědi, typicky HTML kód stránky. Celá odpověď se označí a přes pravé tlačítko myši se vybere možnost *Přidat odkaz*. Nyní se do stromové struktury požadavků vloží *Vlastní kód*. V pravé části okna se třída vlastního kódu pojmenuje a je možné přidat argumenty, kde se vybere dříve vytvořený odkaz. Kliknutím na tlačítko *Generovat kód* se vygeneruje Java třída, která obsahuje metodu `public String exec(ITestExecutionServices tes, String[] args)`, která se bude spouštět.

Samotné nahrazení dat se poté provádí pomocí tlačítka *Substituce*, kde se vybere zdroj dat z možností popsaných výše. Nakonec se u požadavků nastaví tzv. *ThinkTime*, čili čas, jak dlouho se čeká, než se zavolá další požadavek. Nyní jsou uloženy skutečné hodnoty podle toho, jak byl test zaznamenáván. Je dobré tyto hodnoty snížit, aby byl průběh testů rychlý. Ve svých testech jsem tento čas nastavoval většinou v rozmezí jedné až dvou sekund. V případě velkého počtu uživatelů pak ve stovkách milisekund.

Pokud je potřeba upravit průběh testů, lze do stromové sktruktury přidat prvky jako cykly, podmínky a další.

Po veškerých úpravách testovacího skriptu, je vhodné daný skript otestovat, zda projde bez chyby a jeho chování odpovídá očekávání. K otestování lze využít možnost ladit test, která zobrazuje jednotlivé akce testu. Ladění testu je vhodné i pro otestování vloženého vlastního kódu.

Po odladění testu se přidá do projektu *Schedule*. Zde se nastaví požadovaný počet skupin uživatelů a počet uživatelů v jednotlivých skupinách. Každé skupině je možné přiřadit různé testy a také nastavit, jestli se mají spustit okamžitě nebo se zpožděním. I zde lze přidat řídící bloky jako např. cyklus. Pro každou skupinu uživatelů je možné nastavit stroj, ze kterého se má zátěž spouštět. V nastavení plánu lze přidat i možnosti monitorování, jak bylo popsáno v předchozí části.

### 3.3.4 Výhody a nevýhody

+ kvalitní vestavěný záznamník *Recorder*
+ automatické hlídání dynamických částí
+ mnoho možností ovlivnit chování testů včetně doplnění vlastního kódu
+ možnost rozložení zátěže
+ dobré výstupy z testů a monitorování zdrojů
+ ladění testů
+ vestavěná nápověda

- placený nástroj
- větší systémové nároky
- potřeba dalších licencí pro větší počet testovaných uživatelů

## 3.4 Přehled nástrojů pro výkonnostní testování

Tabulka 3.1: *Srovnání nástrojů pro výkonnostní testování*

| Nástroj | Licence | Ladění | Větší možnost úprav skriptů | Práce s nástrojem | Monitorování výkonu | Existence tutoriálů | Celkové hodnocení |
|---|---|---|---|---|---|---|---|
| **JMeter** | open-source | ne | ano | průměrná | průměrné | diskuze na internetu | 4/5 |
| **Eclipse TPTP** | open-source | ne | ne | špatná | špatné | nejsou | 1/5 |
| **IBM RPT** | placený | ano | ano | dobrá | dobré | vestavěná nápověda | 5/5 |

## 3.5 Selenium

### 3.5.1 Popis

Selenium je balík open-source nástrojů, které umožňují automatizovat prohlížeč. Tato funkce umožňuje vytvářet automatizované funkční testy. Selenium poskytuje v podstatě dvě varianty. Tou první je Selenium IDE, což je plugin do prohlížeče Firefox, který umožňuje zaznamenávat a přehrávat testovací scénáře. Druhou možností je Selenium WebDriver, což je balík knihoven pro různé programovací jazyky. Pomocí těchto knihoven je možné programově ovládat prohlížeč. Selenium IDE je nástroj napsaný v jazyce JavaScript a lze jej použít na různých platformách. Selenium WebDriver poskytuje knihovny např. pro jazyky Java, C#, Perl, PHP, Python a další. [8]

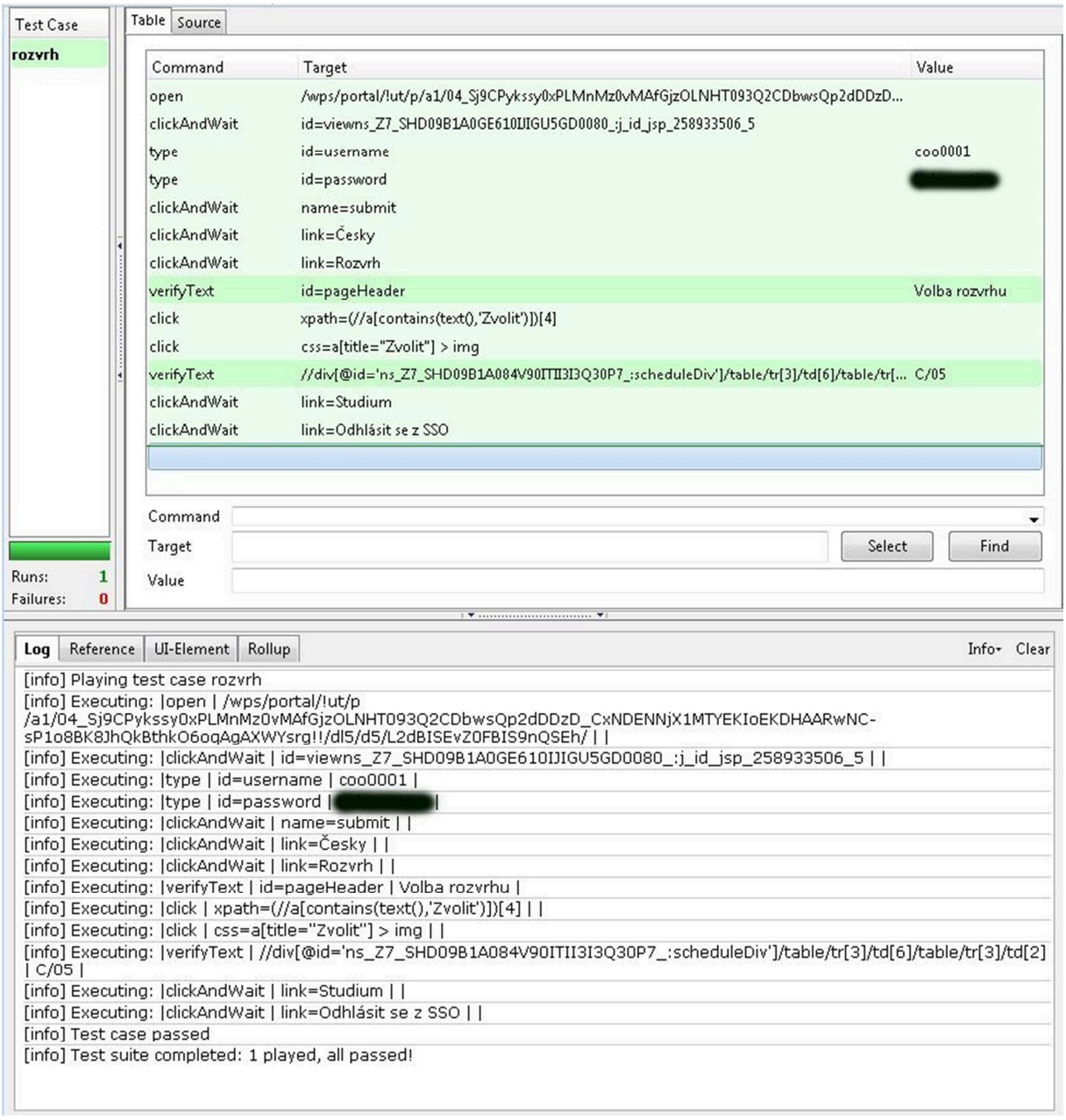
Selenium původně vzniklo jako interní nástroj ve společnosti ThoughtWorks, kde bylo vyvinuto Jasonem Hugginsem v roce 2004. Poté, co tento nástroj předvedl svým kolegům, bylo rozhodnuto, že Selenium budou dále rozvíjet a uvolní jej jako open-source. K uvolnění došlo ještě v roce 2004. Aktuálně je Selenium IDE dostupné ve verzi 2.8 ze září roku 2014. [11]

### 3.5.2 Princip a práce s nástrojem

Selenium IDE se instaluje jako standardní rozšíření prohlížeče Firefox. Program se spouští přímo z prohlížeče. Okno programu je rozděleno na dvě části. V levé části je seznam testovacích scénářů, které se dají spouštět za sebou, a v pravé části jsou jednotlivé příkazy scénáře. Tyto příkazy jsou definovány jazykem Selenese a každý příkaz má dva argumenty: *Target* a *Value*. Příkazy se vytváří buď automaticky zachytáváním akcí uživatele v prohlížeči, nebo manuálně. Jazyk Selenese poskytuje velké množství příkazů, které slouží jak k ovládání prohlížeče, tak k ověřování

a vyhodnocování jednotlivých parametrů a částí webových stránek. Ke každému příkazu aplikace poskytuje krátkou nápovědu, co má být v parametrech vyplněno.

Během provádění testu aplikace automaticky zaznamenává jednotlivé události do Logu, pomocí kterého je pak možné test vyhodnotit.




Obrázek 3.4: *Ukázka výstupu z programu Selenium IDE*

### 3.5.3 Postup vytváření testů

Po zapnutí nástroje se automaticky zapíná zachycování chování v prohlížeči, nahrávání je proto třeba vypnout a nejprve najet na stránky, které chceme testovat. Nyní se nahrávání zapne a začne se zaznamenávat test. Pokud se má po nahrané akci zkontrolovat nějaký prvek, je lepší nahrávání přerušit a manuálně přidat danou kontrolu než nahrát celý test a potom postupně kontroly přidávat.

V případě manuálního přidávání lze využít interaktivní selektor, který pomůže vybrat prvek na stránce a automaticky si ho uloží. Není tedy nutné se po nahrání testu vracet k jednotlivým stránkám. Program umožňuje do seznamu příkazů vkládat i komentáře, což napomáhá k lepší přehlednosti a zjednoduší pozdější úpravu testovacího skriptu.

Po kompletním vytvoření testu se může správnost ověřit spouštěním jednotlivých akcí zvlášť nebo lze spustit celý test a v prohlížeči kontrolovat jednotlivé akce. Po skončení průběhu testu je v logu zaznamenán jeho celý průběh, ze kterého test a jeho úspěšnost vyhodnotíme.

### 3.5.4 Výhody a nevýhody

+ open-source nástroj
+ jednoduché ovládání

− pouze pro prohlížeč Firefox
− pouze pro jednodušší testy

## 3.6 IBM Rational Functional Tester

### 3.6.1 Popis

Jedná se o další aplikaci z komerčního balíku nástrojů určených pro testování, které vyvíjí společnost IBM. Nástroj je určen k automatizovaným funkčním testům. Aplikace je založena na vývojovém nástroji Eclipse a je dostupná pro operační systémy Windows a Linux. Nástroj umožňuje testovat jak webové aplikace, tak také aplikace v prostředí .Net, Java, SAP a další. [10]

Nástroj byl poprvé vydán v létě 2002 pod názvem RobotJ pod záštitou firmy Rational Software, kterou v roce 2003 odkoupilo IBM. Aktuální verze, ve které jsem testy tvořil, je verze 8.6, která vyšla na jaře roku 2014. [11]

### 3.6.2 Princip a práce s nástrojem

Pro záznam testů aplikace používá vestavěný záznamník *Recorder*, který zachytává veškeré uživatelovy akce – klik myší, vstupy z klávesnice. *Recorder* umí při nahrávání rozpoznat jednotlivé prvky aplikace a dané kroky. Jednotlivé akce pak dokáže vztáhnout k daným prvkům např. na základě jejich id. Pokud nástroj nerozezná jednotlivé prvky aplikace, tak potom zaznamená dané kroky s polohou kurzoru myši v podobě souřadnic x a y. První varianta je samozřejmě výhodnější, protože test není závislý na přesné pozici a velikosti okna a je stále funkční i po drobných grafických úpravách aplikace. Pokud se ovšem má testovat např. flashová aplikace běžící na webových stránkách, tak nezbyde nic jiného než se spolehnout na souřadnice.

Veškeré rozpoznané objekty se ukládají do tzv. *Mapy testovacích objektů*, kde jsou následně vidět jednotlivé prvky stránky a jejich vlastnosti. Tyto prvky je možné přidávat do testovacího skriptu a pracovat s nimi. Pokud *Mapa testovacích objektů* nezachytila během nahrávání nějaký objekt, lze jej přidat, a to pomocí vyhledávače objektů, který se aplikuje přetažením kurzoru na daný prvek v prohlížeči.

Po ukončení nahrávání aplikace převede jednotlivé zachycené kroky do Java zdrojového kódu, do jedné metody `public void testMain(Object[] args)`. Zde je možné kód upravit tak,

aby byl efektivnější a přehlednější – např. vyplňování textových a dalších polí je realizováno jako simulace mačkání jednotlivých kláves, ale data je možné vložit pomocí metody `setText()`.

Pro ověřování správného chování testované aplikace se používají tzv. verifikační body. Ty je možné vkládat během nahrávání za pomoci grafického rozhraní anebo po ukončení nahrávání přímo v kódu. Pomocí verifikačních bodů se mohou ověřovat např. zobrazovaná data nebo vlastnost nějakého prvku (např. definovaný CSS styl). V kódu lze manuálně vytvořit verifikační bod následovně:

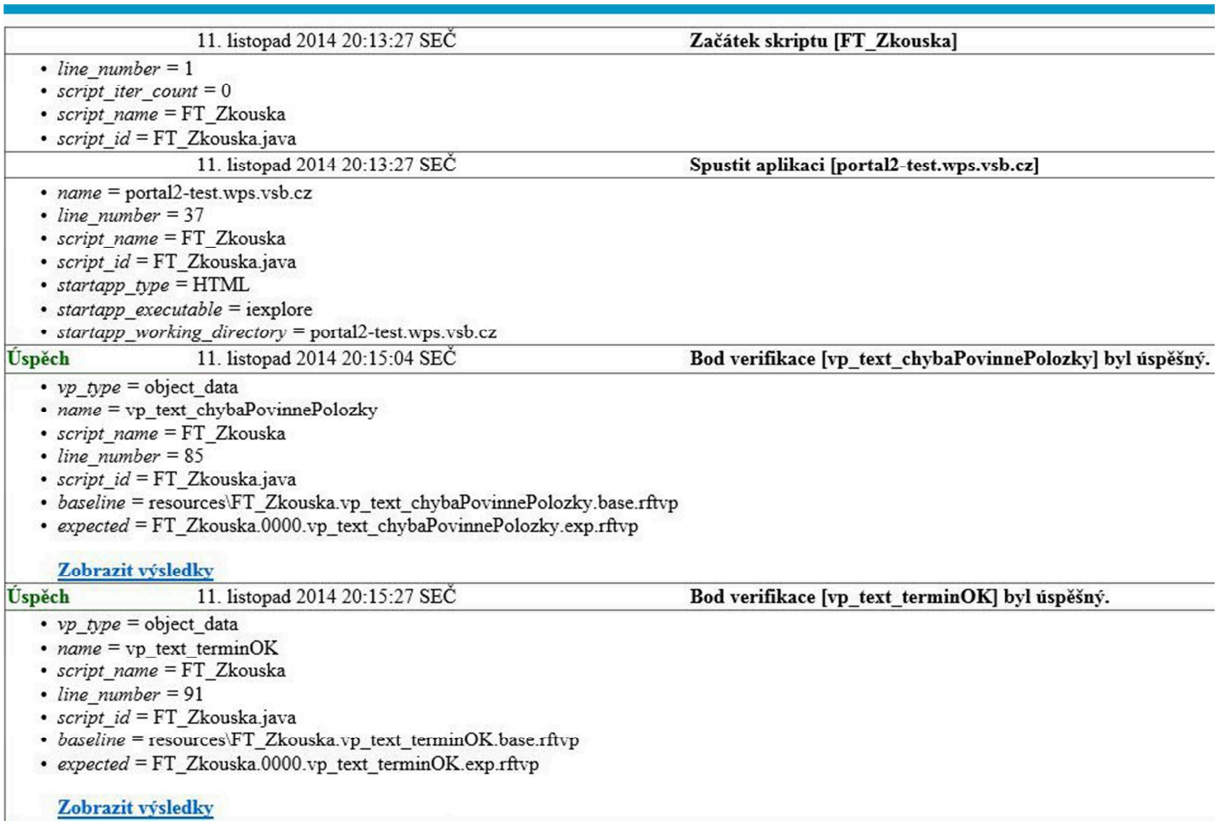
```
IFtVerificationPoint vp = vpManual("name", true, result);
vp.performTest();
```

Parametry metody `vpManual` jsou následující: první parametr určuje název verifikačního bodu, pod jakým má být uložen ve výsledné zprávě, druhý parametr značí očekávaný výsledek a třetí parametr aktuální hodnotu, která je ověřována.

IBM Functional Tester dále obsahuje podobně jako IBM Performance Tester tzv. *Datapool*. Ten umožňuje vytvářet tzv. *data-driven testy*, tedy testy, které jsou spouštěny několikrát za sebou, ale pokaždé s jinými daty. Data do úložiště *Datapool* se mohou buď vyplnit ručně, nebo lze využít importu např. z CSV souboru. Samotná práce s úložištěm *Datapool* je možná dvěma způsoby. Buď je možné v kódu volat odpovídající metody, které vytáhnou data z úložiště *Datapool* podle klíče a při spuštění testu nastavit kolikrát se má daný test spustit a aplikace pak sama iteruje přes jednotlivá data (řádky) úložiště *Datapool*. Druhou možností je nastavit spuštění testu pouze jednou, ale v kódu manuálně daný *Datapool* připojit a ve smyčce jej procházet a získávat odpovídající data.

Po spuštění testovacího skriptu nástroj automaticky spustí danou aplikaci a začne vykonávat jednotlivé úkony. Během vykonávání testu se nesmí hýbat s myší, protože se při testování simuluje její pohyb a jednotlivá kliknutí. Po skončení běhu skriptu se vygeneruje protokol o testu ve formátu HTML, který obsahuje jednotlivé verifikační body s jejich daty a výsledkem, zda byl bod úspěšný nebo ne. Současně jsou zde zaznamenány i chyby testu – např. objeví-li se neošetřená výjimka apod.

Při práci s nástrojem se také někdy může stát, že se přestane test správně nahrávat, aplikace se začne zasekávat, nebudou se identifikovat objekty apod. Tyto problémy lze většinou odstranit tím, že se aplikaci smažou metadata, která si automaticky vytváří ve svém *Workspace*.




Obrázek 3.5: *Ukázka výstupu z programu IBM RFT*

### 3.6.3 Postup vytváření testů

Pro správnou funkčnost programu je potřeba jej spouštět jako správce. Před samotným vytvářením testů je nutno nadefinovat aplikaci, kterou budeme testovat. V menu *Konfigurovat aplikace pro testování* se vybere možnost *Přidat novou aplikaci*. Pro testování webové aplikace se zvolí *Aplikace v jazyce HTML* a zadá se URL, kde daná aplikace běží. Zvolí se ještě, jaký prohlížeč se má pro nahrávání použít. Doporučeným prohlížečem je Internet Explorer, funkční je i Mozilla Firefox, ale je potřeba vždy pro danou aplikaci ověřit, zda se nahrávání testu v použitém prohlížeči správně zaznamenává.

Po nastavení aplikace se vytvoří nový projekt typu *Projekt testu funkčnosti*. Pokud se v testovacím skriptu má používat *Datapool*, je třeba jej do projektu vložit a naplnit daty. Při vyplňování dat do úložiště *Datapool* je vhodné také zahrnout proměnnou, která nám označí, zda se jedná o data s očekávaným kladným výsledkem nebo o chybu.

Nyní se do projektu vytvoří samotný skript pomocí možnosti *Přidat skript pomocí záznamníku*. Zde je důležité ověřit, že je pro režim záznamu vybráno *Skriptování v jazyce Java*. Automaticky se otevře dialogové okno, pomocí kterého se může nahrávání testu ovládat. Vybere se možnost *Spustit aplikaci*, čímž se otevře okno, označí se aplikace, která byla nadefinována na začátku. Nutno podotknout, že nesmí být otevřené žádné jiné okno prohlížeče, který byl zvolen pro nahrávání testu. Nyní se mohou v aplikaci procházet jednotlivé kroky, které se mají testovat. Pro vložení verifikačního bodu se použije dialogové okno, které umožní pomocí *Vyhledávače objektů* vybrat

a identifikovat v testované aplikaci položku, nad kterou se má provést ověření. Zvolí se, jaká vlastnost se má ověřovat a data, která jsou očekávána.

Při vyplňování např. formulářů v dané aplikaci lze použít dříve vytvořený *Datapool*, ze kterého se vyberou doplňovaná data. Během nahrávání je vhodné sledovat jednotlivé akce, které jsou zaznamenávány a vypisovány v dialogovém okně pro ověření, zda se test nahrává správným způsobem a také, že se správně rozeznávají jednotlivé objekty, tzn., že se nezaznamenávají pouze souřadnice, kde bylo kliknuto myší. Před samotným ukončením nahrávání je vhodné, jako poslední krok testu, zařadit akci ukončení prohlížeče, aby se poté dané testy daly spouštět za sebou bez nutnosti manuálního ukončování prohlížeče. O přehrávání testu platí totiž to stejné, co u nahrávání – nesmí být otevřeno žádné jiné okno daného prohlížeče, jinak aplikace neví, v které instanci má jednotlivé kroky provádět. Samotné ukončení nahrávání testu se provede přes ovládací ikonu z dialogového okna záznamníku *Recorder*.

Nyní se zobrazí kód v jazyce Java, který definuje jednotlivé kroky testu. Je potřeba tyto kroky zkontrolovat a upravit. Podle toho, co se má testovat, lze doplnit například cykly, manuální verifikační body atd. Některé kroky je lepší a efektivnější přímo doprogramovat než definovat během nahrávání. Např. to může být situace, kdy se má ověřovat obsah tabulky. Pro takovou situaci je nejlepší během nahrávání pouze identifikovat objekt tabulky (např. pomocí vytvoření pomocného verifikačního bodu, který se potom nepoužije). V kódu se pak z objektu načtou jednotlivé buňky, přes které se bude v cyklu iterovat a kontrolovat data. Je zde také možné upravit, která data se budou brát z úložiště *Datapool*, případně úplně manuálně načíst *Datapool* a iterovat přes jednotlivé hodnoty.

Pokud je celý testovací scénář připraven, může se test spustit. Během přehrávání testu se otevře nadefinovaný prohlížeč a začnou se provádět jednotlivé akce. Někdy se může stát, že se test nepřehrává správně, např. že se mu nedaří nalézt a identifikovat jednotlivé objekty. Toto chování lze napravit pomocí nastavení, kde lze definovat jednotlivé časy, jak velké mají být prodlevy mezi simulací kliků myši, stisknutí kláves, jak velký má být maximální čas pro hledání objektů apod.

### 3.6.4 Výhody a nevýhody

+ kvalitní vestavěný záznamník *Recorder*
+ mnoho možností jakkoliv upravit skript v kódu
+ dobrá práce s daty pro testování – úložiště *Datapool*

− placený nástroj
− větší systémové nároky
− občas podivné chování nástroje

## 3.7 Přehled nástrojů pro automatizované funkční testování

Tabulka 3.2: *Srovnání nástrojů pro automatizované funkční testování*

| Nástroj | Licence | Ladění | Možnost složitějších skriptů | Práce s nástro-jem | Ověření funkčnosti | Existence tutoriálů | Celkové hodno-cení |
|---|---|---|---|---|---|---|---|
| **Selenium** | open-source | ne | ne | průměrná | průměrné | minimální | 2/5 |
| **IBM RFT** | placený | ano | ano | dobrá | dobré | vestavěná nápověda | 4/5 |

## 3.8 Zhodnocení testovacích nástrojů

Pro automatizované a výkonnostní testování existuje velké množství nástrojů. Představil jsem a popsal některé z nich, a to jak placené nástroje, tak open-source nástroje. Většina těchto nástrojů funguje na podobném principu nahrání základní kostry testu a jejím následném manuálním upravení. Dle mých poznatků o těchto nástrojích, bych doporučil nástroje od firmy IBM, čili Rational Performance Tester a Rational Functional Tester. Jedná se ovšem o placené nástroje, které lze bez licence používat jen po omezenou zkušební dobu. Z open-source nástrojů bych doporučil program JMeter, který má velké možnosti a také lze pro něj nalézt spoustu návodů a tutoriálů.

Pro vytváření testů v rámci této diplomové práce jsem zvolil výše uvedené produkty společnosti IBM.

# 4 Popis infrastruktury systému Edison

## 4.1 Produkční prostředí

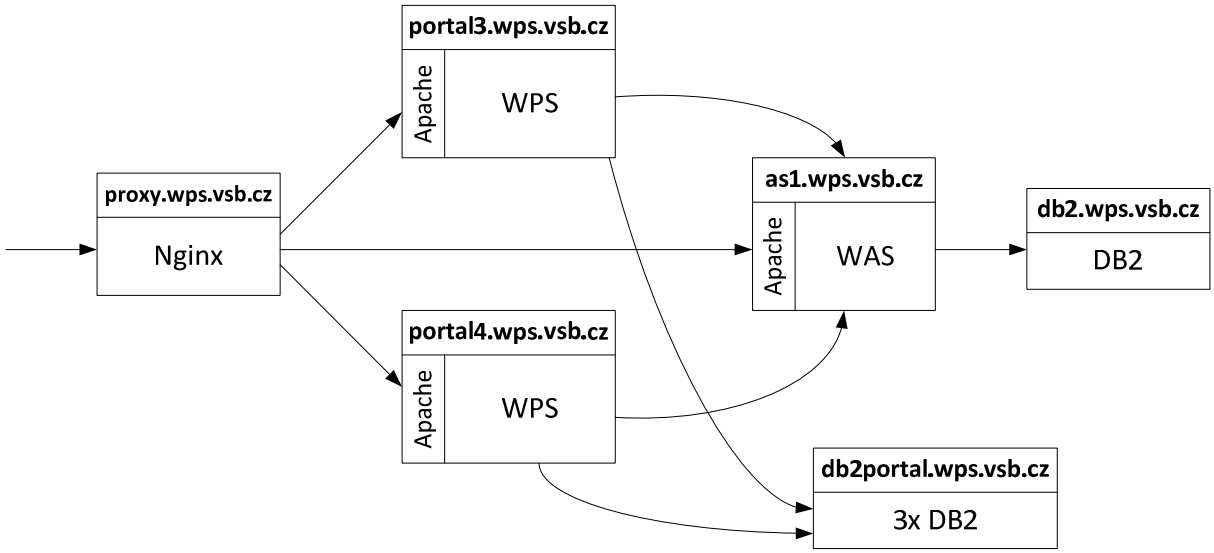
Infrastruktura informačního systému Edison se skládá z několika komponent a veškeré servery běží ve virtuálním prostředí Centra informačních technologií na Vysoké škole báňské – Technické univerzitě Ostrava.

Na začátku řetězce je proxy server *proxy.wps.vsb.cz*, na kterém běží služba Nginx, která na základě IP hashe předává požadavky na dva portálové servery *portal3.wps.vsb.cz* a *portal4.wps.vsb.cz*. To znamená, že ze stejné IP adresy se budeme vždy připojovat na stejný portálový server, za předpokladu, že oba dva servery poběží. Tyto dva portálové servery jsou identické, ale neběží v clusteru. Na portálových serverech běží Apache a WebSphere Portal Server (WPS) od společnosti IBM. Právě na Apache se předávají požadavky z Nginx. Tyto portálové servery se starají o prezentační část informačního systému. Dále se dotazují na aplikační server *as1.wps.vsb.cz*, na kterém běží také Apache a dále WebSphere Application Server (WAS). Portálové servery se dotazují přímo na WAS, tzn. mimo Apache. Další možností dotazu na aplikační server je dotaz přímo z proxy. V takovém případě jdou požadavky na Apache aplikačního serveru a pak na WAS.

Součástí infrastruktury jsou i dva databázové servery. Prvním je *db2portal.wps.vsb.cz*, na kterém běží tři instance databáze DB2 od IBM. První dvě instance jsou pro produkční portálové servery *portal2.wps.vsb.cz* a *portal3.wps.vsb.cz*. Poslední třetí instance je pro testovací prostředí. V každé instanci je 6 databází, kde se ukládají podpůrné informace pro portálové servery a které WPS potřebují pro svou funkčnost.

Druhým databázovým serverem je *db2.wps.vsb.cz*, kde běží jedna instance databáze DB2 od IBM. V této instanci je jedna databáze s názvem OSTRA obsahující data celého informačního systému Edison. Tato databáze v současnosti obsahuje desítky schémat a stovky tabulek.

Je nutno podotknout, že ačkoliv celá síťová infrastruktura na VŠB-TUO běží jak na IPv4, tak na IPv6, systém Edison je přístupný pouze přes IPv4.

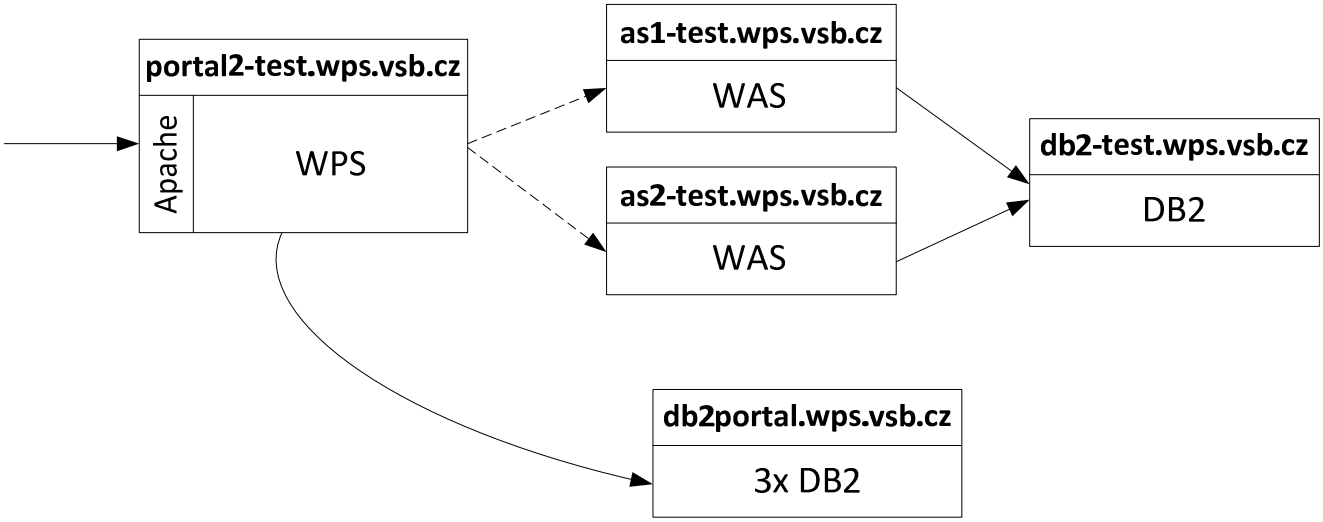



Obrázek 4.1: *Produkční infrastruktura systému Edison*

## 4.2 Testovací prostředí

Testovací infrastruktura obsahuje pouze jeden portálový server *portal2-test.wps.vsb.cz*, na kterém také běží Apache a WPS. Dále je zde aplikační server *as1-test.wps.vsb.cz* s WAS, který má ovšem přidělenou podstatně nižší paměť než produkční aplikační server. Přidání další paměti by nepomohlo, protože na serveru je nainstalován WAS ve verzi Express, která je pouze 32 bitová. Proto pro účely výkonnostních testů vznikl nový aplikační server *as2-test.wps.vsb.cz*, který již odpovídá svou konfigurací produkčnímu aplikačnímu serveru.

Poslední částí je databázový server *db2-test.wps.vsb.cz* s jednou instancí databáze DB2. Běží zde ovšem dvě databáze, a to databáze TEST a databáze POKUSNA. Obsah těchto dvou databází je kopií ostré databáze, s tím, že kopie se provádí většinou v intervalu tří měsíců. Pro provoz WPS na *portal2-test.wps.vsb.cz* jsou také potřeba podpůrné databáze, které běží ve třetí instanci na serveru *db2portal.wps.vsb.cz*.




Obrázek 4.2: *Testovací infrastruktura systému Edison*

## 4.3 Hardwarová konfigurace serverů

- **proxy.wps.vsb.cz**
  - 2 procesory Intel Xeon CPU E5-2660v2 2,2 GHz
  - 4 GB RAM

- **portal3.wps.vsb.cz a portal4.wps.vsb.cz**
  - 2 procesory Intel Xeon CPU E5-2660v2 2,2 GHz
  - 16 GB RAM

- **as1.wps.vsb.cz**
  - 12 procesorů Intel Xeon CPU E5-2665 2,4 GHz
  - 16 GB RAM

- **db2.wps.vsb.cz**
  - 6 procesorů Intel Xeon CPU E5-2665 2,4 GHz
  - 24 GB RAM

- **db2portal.wps.vsb.cz**
  - 4 procesory Intel Xeon CPU E5-2665 2,4 GHz
  - 16 GB RAM

- **portal2-test.wps.vsb.cz**
  - 2 procesory Intel Xeon CPU E5-2660v2 2,2 GHz
  - 8 GB RAM

- **as1-test.wps.vsb.cz**
  - 12 procesorů Intel Xeon CPU E5-2665 2,4 GHz
  - 6 GB RAM

- **as2-test.wps.vsb.cz**
  - 12 procesorů Intel Xeon CPU E5-2665 2,4 GHz
  - 16 GB RAM

- **db2-test.wps.vsb.cz**
  - 6 procesorů Intel Xeon CPU E5-2665 2,4 GHz
  - 24 GB RAM

## 4.4 Verze služeb na serverech

- **proxy.wps.vsb.cz**
  - OS: CentOS 6.6, 64 bit
  - Nginx 1.6.2

- **portal3.wps.vsb.cz a portal4.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - Apache 2.2.15
  - IBM Java, JRE 1.6.0
  - WebSphere Portal Server 8.0.0.1, který běží v kontejneru WebSphere Application Server Network Deployment 8.0.0.8

- **as1.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - Apache 2.2.15
  - IBM Java, JRE 1.6.0
  - WebSphere Application Server 8.5.5.0

- **db2.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - DB2 Workgroup Server Edition 10.1

- **db2portal.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - DB2 Workgroup Enterprise Edition 10.1

- **portal2-test.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - Apache 2.2.15
  - IBM Java, JRE 1.6.0
  - WebSphere Portal Server 8.0.0.1, který běží v kontejneru WebSphere Application Server Network Deployment 8.0.0.8

- **as1-test.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - IBM Java, JRE 1.6.0
  - WebSphere Application Server Express 8.5.5.0

- **as2-test.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - IBM Java, JRE 1.6.0
  - WebSphere Application Server 8.5.5.0

- **db2-test.wps.vsb.cz**
  - OS: Red Hat Enterprise Linux 6.6, 64 bit
  - DB2 Workgroup Enterprise Edition 10.1

## 4.5 Použité Java technologie

### Prezentační vrstva na portálových serverech

- Portlety JSR 286
- Java Server Faces (JSF) 2.0
- Java Server Pages (JSP)
- kombinace JavaScriptu a AJAX

### Aplikační server

- Enterprise Java Beans (EJB) 3.1
- Java servlety
- Java Server Faces (JSF) 2.0
- Hibernate 3.6 pro mapování datové vrstvy

# 5 Monitorování a sběr dat v systému Edison

Součástí produkčního provozu informačního systému Edison je i monitorování systému. Monitorovací systémy hlídají jednotlivé parametry serverů a aplikací a v případě výjimečných situací posílají upozornění odpovědným pracovníkům oddělení CIT. Současně monitorovací nástroje také ukládají historii vytíženosti a případně dalších parametrů. Tuto historii lze použít k identifikaci potíží se systémem, případně k identifikaci probíhajících bezpečnostních útoků. V současnosti se používají tři monitorovací nástroje nad systémem Edison. Prvním z nich je Hyperic HQ, který je využíván vývojáři a který monitoruje hlavně aplikační parametry. Dále se využívá systému Nagios a Cacti, které monitorují serverové parametry a slouží správcům systémů ze serverového týmu.

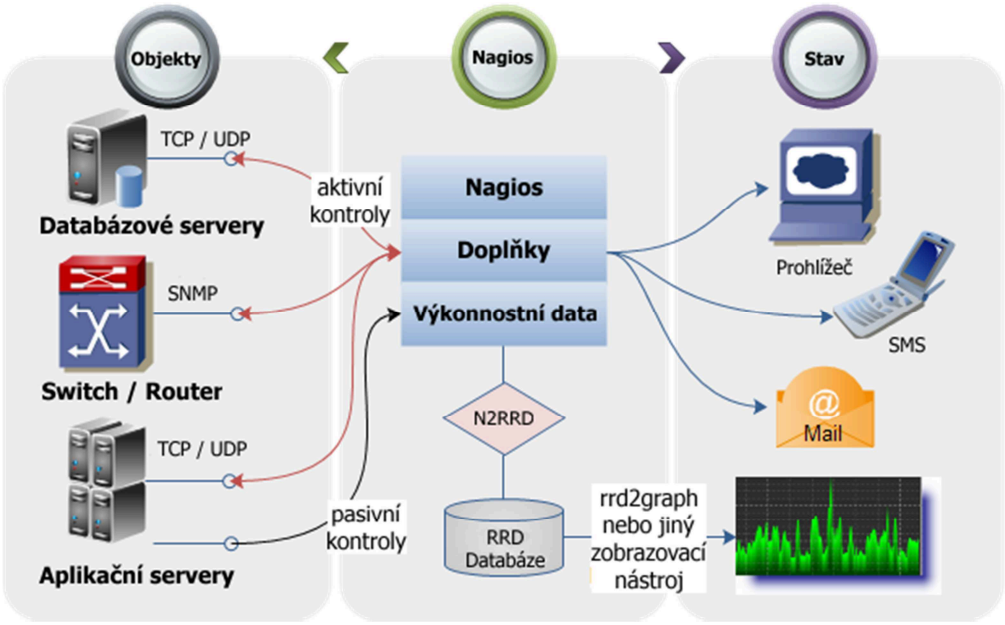
## 5.1 Nagios

### 5.1.1 Popis

Nagios je open-source monitorovací nástroj určený pro Unixové platformy, který umí sledovat různé systémy serverové a síťové infrastruktury. Nagios umožňuje sledovat různé síťové služby jako např. HTTP, SMTP, POP3, FTP a další. Umí také sledovat systémové zdroje, jako využití CPU a pamětí. [18]

Pomocí dostupných pluginů lze monitorovat v podstatě jakoukoliv veličinu. To se provádí za pomoci skriptu, který běží na straně monitorovaného systému a vrací hodnoty 0 až 2. Hodnota 0 znamená stav OK, hodnota 1 stav WARN a hodnota 2 stav CRITICAL. Nagios se potom dotazuje na tyto hodnoty a ukládá je.

Výhodou Nagiosu je to, že umí na základě nastavení rozesílat upozornění, pokud daná služba dosáhne nakonfigurované hodnoty. Tato upozornění jsou odesílána buď emailem nebo za pomoci SMS zpráv.




Obrázek 5.1: *Princip systému Nagios [19]*

### 5.1.2 Využití nástroje u systému Edison

Monitoring Nagios se používá především pro sledování služeb a vlastností, které jsou kritické pro správný běh aplikace. Při překročení nadefinovaných hodnot u těchto služeb se rozesílají upozornění odpovědným pracovníkům oddělení CIT.

Pomocí monitoringu Nagios se u veškerých serverů, které jsou pod správou oddělení CIT, monitorují tyto základní vlastnosti:

- vytížení procesorů,
- využití pamětí,
- velikost SWAPu,
- počet běžících procesů,
- zombie procesy,
- volné místo na pevném disku.

Pro portálový a aplikační server Edisonu se navíc monitoruje dostupnost na portech 80 a 443, tedy protokoly HTTP a HTTPS.

Dotaz na stav jednotlivých služeb probíhá standardně co 5 minut. V případě zjištění kritické hodnoty probíhají dotazy každou minutu, až do opravy problému.

## 5.2 Cacti

### 5.2.1 Popis

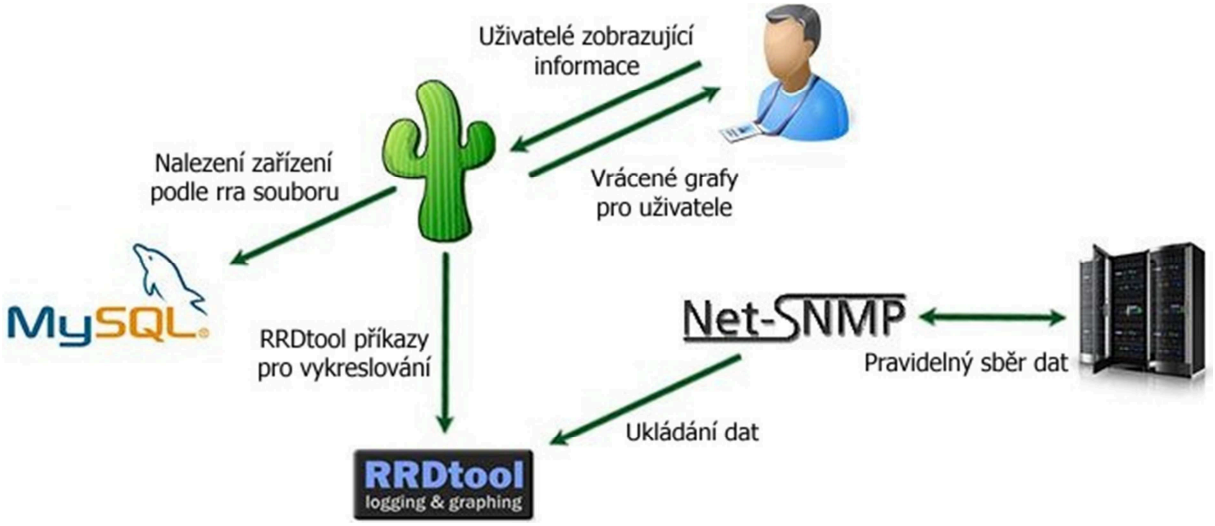
Cacti je opět open-source monitorovací nástroj. Jedná se vlastně o frontend ke sběru dat pomocí nástroje RRDTool. Tento nástroj sbírá nakonfigurované informace pomocí SNMP protokolu z různých systémů a ukládá je do MySQL databáze. Na základě těchto nasbíraných dat umí vykreslovat grafy, které slouží k analýze vytíženosti systému. Frontend této aplikace je napsaný v jazyce PHP. Účelem tohoto monitorovacího nástroje není rozesílat reporty po dosažení kritické hodnoty nějaké služby, ale pouze zaznamenávat stav sledovaných služeb a ukládat jej do databáze tak, aby bylo možné se k těmto údajům vracet. Cacti v sobě obsahuje také správu uživatelů, což umožňuje administrátorům přidělit práva pro sledování systému určitým osobám. [20]

### 5.2.2 Využití nástroje u systému Edison

Systém Cacti se používá na oddělení CIT k pasivnímu monitoringu a slouží hlavně k náhledu historie vytíženosti serverů a pro zpětnou identifikaci příčiny reportovaného problému.

U serverů, které tvoří systém Edison, se monitorují stejné služby a zdroje jako u ostatních serverů pod správou oddělení CIT. Jsou jimi:

- vytížení pamětí – detailní monitoring rozdělený na několik částí,
- vytížení procesorů,
- obsazené místo na pevném disku,
- velikost provozu na jednotlivých síťových rozhraních.




Obrázek 5.2: *Princip systému Cacti [21]*

## 5.3 Hyperic HQ

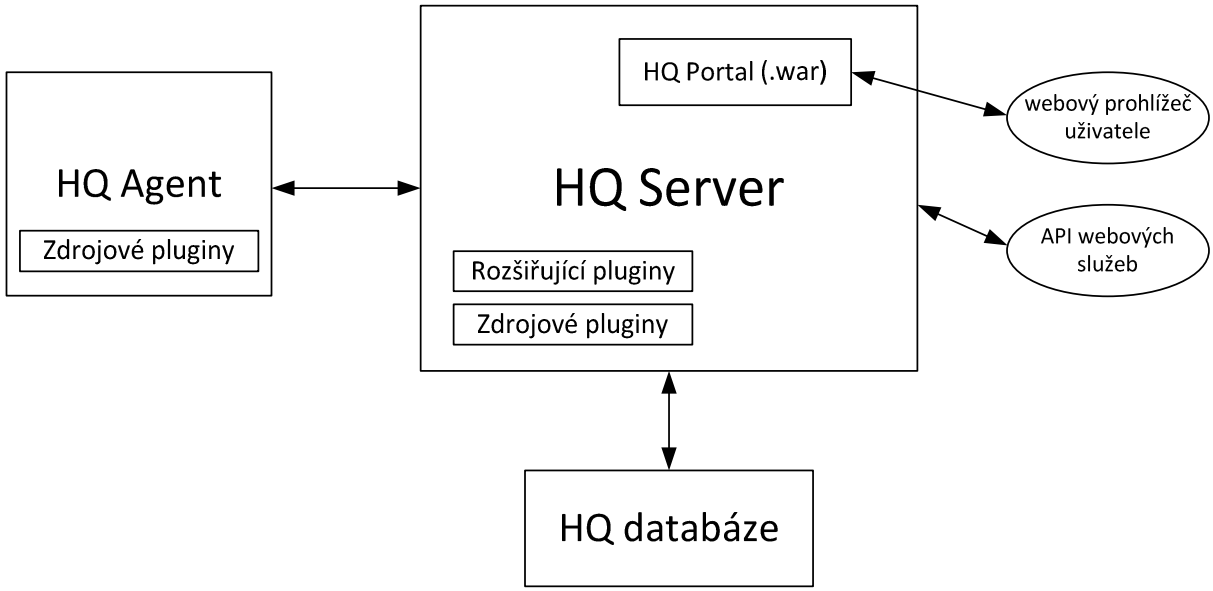
### 5.3.1 Popis

Hyperic je populární open-source monitorovací nástroj, který dokáže sám detekovat systémové zdroje a jejich metriky. Mezi zdroje pro monitorování patří hardware, operační systémy, virtualizované systémy, databáze, middleware, aplikace a služby. Hyperic sleduje nakonfigurované stroje a služby, ze kterých ukládá dané naměřené údaje do databáze a umí také rozesílat notifikace při dosažení nakonfigurované hodnoty. [22]

Celý systém Hypericu se skládá ze tří částí. Tou první je Hyperic Agent. Agent běží na každém stroji, který má být monitorován. Agenti obsahují funkce pro automatickou detekci komponent běžících na monitorovaném stroji a v pravidelných intervalech skenují danou platformu pro zjištění změn v konfiguraci. Agenti získávají a hromadí výkonnostní metriky a posílají je na centrální Hyperic Server, kde se tyto údaje ukládají. Agenti současně umějí provádět i akce na daných strojích, jako může být například start nebo zastavení serverů. [23]

Druhou součástí je Hyperic Server s Hyperic Databází. Server přijímá informace a metriky od jednotlivých agentů a ukládá je do databáze. Server umožňuje spravovat inventář jednotlivých posbíraných dat a sdružovat je do různých skupin pro zjednodušení monitorování a správy. Hyperic Server poskytuje také autentizaci pro jednotlivé služby a detekuje překročení kritických hranic, na základě kterých rozesílá upozornění jednotlivým uživatelům. Na serveru potom běží ještě poslední část Hypericu a to je Hyperic Portal. Jedná se o Java webovou aplikaci, která poskytuje k monitorovacímu nástroji grafické rozhraní přístupné přes webový prohlížeč. Přes toto grafické rozhraní lze celý systém konfigurovat a zobrazovat údaje a grafy ze zaznamenaných dat. [23]

Ukázka propojení jednotlivých částí systému Hyperic je na Obrázku 5.3.




Obrázek 5.3: *Princip systému Hyperic HQ [24]*

Další výhodou systému Hyperic je, že kromě standardních agentů obsahuje i pluginy, které se umí připojit na různé služby jako je například *WebSphere PMI (Performance Monitoring Infrastructure)*. Jiný plugin Hypericu se umí připojit na *JMX (Java Management Extensions)* rozhraní aplikace a je tak možné sbírat v podstatě jakékoliv údaje, které aplikace poskytne.

### 5.3.2 Využití nástroje u systému Edison

Monitorovací nástroj Hyperic se používá na oddělení CIT pracovníky vývojářského týmu. Monitorují se hardwarové hodnoty jako je využití procesorů nebo pamětí. Dále se využívá napojení na monitoring od aplikačního a portálových serverů IBM WebSphere. Současně se využívá také JMX rozhraní k monitorování vlastních věcí v aplikaci Edison. Na aplikačním serveru jsou Java Beany, které poskytují data o volbě rozvrhu a volbě sportů.

Využití Hypericu je jak pro pasivní monitoring a zjišťování vytíženosti v historii, tak pro rozesílání notifikací po překročení kritické meze. Nevýhodou Hypericu je, že provádí dotazy na stav služeb s nejkratším možným intervalem jedné minuty, a proto se nedá použít pro sledování zátěže v reálném čase během výkonnostních testů. Pro takový případ je nutné použít přímo administrační konzoli IBM WebSphere.

## 5.4 Zhodnocení monitoringu nad systémem Edison

Informační systém Edison je v současnosti dostatečně pokryt monitorovacími nástroji, které v případě výpadku nebo jiné neobvyklé situace okamžitě informují pověřené pracovníky. Zároveň také umožňují vyhodnocovat dlouhodobé vytížení jednotlivých serverů a služeb. Pokryta je hardwarová i softwarová část systému. Případné rozšíření monitorování by bylo vhodné v případě vzniku nové části systému, u které by se očekávala velká zátěž generovaná koncovými uživateli. V takovém případě by bylo vhodné poskytovat z aplikace Edison monitoringu Hyperic aplikační statistiky, na základě kterých by se daly identifikovat příčiny případných problémů.

Nutno také podotknout, že veškeré zde popsané monitorování pokrývá produkční část systému a nad testovacím prostředím monitoring zaveden není. Proto při monitorování vytížení během testů je potřeba sledovat zátěž v reálném čase např. pomocí nástroje *htop* nad fyzickými servery a pomocí monitorovacího rozhraní v administraci IBM WebSphere.

# 6 Metodika testování pro informační systém Edison

Pro usnadnění vytváření testů a samotného procesu testování, a to jak automatizovaného funkčního testování, tak výkonnostního testování, slouží níže popsaná metodika. Tato metodika je přímo zaměřena na informační systém Edison a testovací nástroje od společnosti IBM a vychází z obecných informací uvedených v [2, 3].

## 6.1 Příprava testovacího prostředí

Pro vytváření, ladění a vykonávání automatizovaných funkčních a výkonnostních testů je zapotřebí mít k dispozici testovací prostředí, které bude oddělené od produkčního prostředí. Oddělit je potřeba kromě jednotlivých serverů i databázi. Pro informační systém Edison již existuje testovací prostředí, které je využíváno vývojáři a dalšími pracovníky oddělení CIT pro základní manuální testování, analýzu nových funkčností, simulace různých situací, které pomáhají při řešení problémů koncových uživatelů, a také třeba pro školení odpovědných pracovníků. Pro tyto popsané úlohy se používá databáze TEST, která obsahuje kopii dat databáze OSTRA.

Pro potřeby výkonnostního testování byla hardwarová konfigurace portálového serveru povýšena tak, aby odpovídala konfiguraci používané na produkci. Dále vznikl nový aplikační server *as2-test.wps.vsb.cz* s hardwarovou konfigurací, která opět odpovídá produkci. Tento nový aplikační server pracuje s databází POKUSNA a nepoužívá se pro běžný provoz testovacího prostředí, které je popsáno v předchozím odstavci. Důvodem částečného oddělení testovacího prostředí pro běžný provoz od testovacího prostředí pro výkonnostní testy, je velký zásah do databáze pro přípravu testovacích scénářů. Ideálním stavem by bylo úplně samostatné testovací prostředí, které by bylo určené pouze pro výkonnostní a případně automatizované funkční testy.

Z důvodu sdílení testovacího prostředí pro více účelů a používání dvou oddělených aplikačních serverů, je potřeba udržovat verze systému Edison na obou aplikačních serverech stejné. Pro vytváření a ladění testů zpravidla není potřebný až tak velký zásah do databáze a tyto úkony lze provádět paralelně s běžným provozem na testovacím prostředí, tzn., že nedochází k omezení ostatních zaměstnanců. Příprava databáze probíhá zpravidla přímo pomocí SQL příkazů a není tedy závislá na aktuální konfiguraci testovacích serverů.

Pro spouštění testů je potřeba upravit testovací prostředí tak, aby portálový server pracoval s druhým aplikačním serverem *as2-test.wps.vsb.cz*. Dále je potřeba také drobně upravit vzhled aplikace pro výběr studijních vztahů studenta. Tento výběr probíhá za normálních okolností z rolovací nabídky, ale při velkém počtu studijních vztahů by načítání tohoto rolovacího menu mohlo ovlivnit objektivitu výkonnostního testu, a proto je rolovací menu nahrazeno obyčejným textovým polem, do kterého se vkládá ID studijního vztahu.

Další drobnou komplikací, která plyne ze sdílení testovacího prostředí pro více účelů, je potřeba domluvit čas spouštění a provádění testů tak, aby v tomto čase k systému nikdo jiný nepřistupoval.

## 6.2 Identifikace částí systému pro testování

### 6.2.1 Automatizované funkční testy

Informační systém Edison je funkční systém, který běží již několik let. Pravidelně každé dva týdny je vydávána nová verze, která se v úterý nahrává na testovací prostředí a v pátek probíhá nahrání na produkční servery. V období po nahrání na testovací servery probíhá ověřování správného chování nových funkčností pracovníky oddělení CIT pomocí manuálních testů. Zde je právě prostor pro implementaci automatizovaných funkčních testů, ale je potřeba správně identifikovat rozsah nové funkčnosti tak, aby implementace automatizovaného testu nebyla složitější a mnohem časově náročnější než provedení manuálního testování.

Dalším uplatněním pro automatizované funkční testy by mohly být regresní testy, kde by se několika testy pokryla velká množina funkčností systému a vždy při testování nové verze, by se z množiny regresních testů vybraly ty, které testují související funkčnosti. Tento způsob testování by umožňoval automatické ověřování zachování funkčnosti současných vlastností po nasazení nové verze.

### 6.2.2 Výkonnostní testování

Identifikace částí systému pro výkonnostní testování plyne hlavně z dlouhodobějšího monitorování a sledování vytížení systému. Nejedná se o činnost, ke které by docházelo v pravidelných intervalech s nasazováním nové verze systému. Výjimkou by bylo nasazení nové funkčnosti, u které se očekává velké zatížení, případně oprava či změna částí systému, které byly dříve identifikovány a určeny k výkonnostnímu testování.

Zdrojem informací o tom, že je potřeba danou část systému otestovat může být výstup monitorování systému nebo zaznamenané chyby v logu aplikace, případně v logu serverů. Z těchto informací se zjistí většinou jenom čas události, ale hlavní je identifikovat, kde a proč daná zátěž vznikla. To znamená, že je také potřeba vědět, co se v danou dobu dělalo a jaké akce byly spuštěny. Zde může patřit např. zápis do rozvrhu, volba sportu, přihlašování na zkoušky a další. Dalším možným zdrojem informací je potom hlášení koncových uživatelů o pomalé odezvě systému při daných činnostech.

## 6.3 Znalost dané části systému pro testování

Pro návrh testovacího scénáře a následné vyhodnocování testu, je potřeba mít dostatečnou znalost dané části systému. Pro funkční testování jde zejména o znalost chování systému, vyhodnocování jednotlivých situací, vyhodnocování daných vstupů apod. Tyto znalosti lze zpravidla nabýt ze specifikace zadání pro danou funkčnost systému.

Pro výkonnostní testování je potřeba trochu jiných znalostí než pro testování funkční. Samozřejmě i zde je potřeba znát chování systému a vědět, jak se zachová v jaké situaci. Důležitá je zde znalost parametrů jednotlivých HTTP požadavků, tzn., co se jaké stránce předává za parametry, co znamenají, jaký mají dopad a kde se berou. Další důležitou věcí pro přípravu testovacích scénářů je znalost chování uživatelů v dané sekci systému. Tato znalost se dá získat nejlépe od koncových uživatelů. V případě, že to není možné, je nejlepší si vyzkoušet danou situaci sám za pomoci nasimulování konkrétního úkolu.

Společným faktorem pro funkční i výkonnostní testování je identifikace a znalost nutných prerekvizit (předpokladů) pro daný test. Ne vždy může být úplně jednoduché identifikovat a přesně definovat všechny prerekvizity, které musejí být splněny. Při zjištění potřebných prerekvizit je vhodné začít zkoušením funkčnosti systému v různých situacích. Tím ovšem nemusí být odhaleny všechny možnosti a je potřeba ještě prozkoumat specifikaci dané funkčnosti, případně uživatelskou dokumentaci.

## 6.4 Příprava prerekvizit pro daný test

První typickou prerekvizitou, kterou je potřeba vyřešit je přístup uživatelů do systému. Na testovacím portálovém serveru je z důvodu licence možné povolit pouze omezený počet uživatelů. Současně s tím souvisí i další problém, a to je neexistence testovacího LDAP systému. Proto byl v ostrém LDAP systému vytvořen jeden testovací uživatel s osobním číslem *coo0001*, kterému byl přístup povolen. V případě potřeby přístupu dalšího uživatele je nutné tuto skutečnost také nastavit. Zároveň je těmto uživatelům třeba nastavit platná hesla tak, aby se byli vůbec schopni do systému přihlásit.

Vzhledem k tomu, že jsme při testování omezeni pouze na jednoho testovacího uživatele, ale při výkonnostních testech je potřeba spouštět několik uživatelů současně, bylo nutné tuto situaci vyřešit a obejít jiným způsobem. Řešením bylo tomuto jednomu uživateli přiřadit požadovaný počet studijních vztahů. Z této množiny si poté uživatel vždy na začátku testu jeden vybere. Tím je docíleno simulace více uživatelů pracujících současně s odlišnými daty.

Další prerekvizity plynou z požadavků samotných testů. Některé prerekvizity je možné nastavit přímo přes uživatelské rozhraní. Toho se využívá, pokud se jedná o menší změnu u jednoho uživatele. Typicky se jedná např. o situaci kdy je potřeba uživateli přiřadit roli vyučujícího apod. Toto nastavení lze udělat ideálně pomocí uživatele, který má v systému nastavená administrátorská práva. U testovacího serveru se jedná např. o vývojáře systému.

Hromadné změny a větší úpravy je efektivnější provádět přímo v databázi pomocí SQL příkazů. K takovým změnám může patřit např. přiřazování studijních vztahů testovacímu uživateli, nastavení stavu zapsaných předmětů, úprava termínů pro volbu rozvrhů, sportů, osobních studijních plánů a další.

Pro ukázku nastavení prerekvizit může sloužit např. SQL příkaz, který provede přiřazení všech studijních vztahů z fakulty FEI, prezenčního bakalářského a magisterského studia uživateli *coo0001*. Úprava se nedotkne vyjmenovaných uživatelů, kteří jsou zaměstnanci oddělení CIT, aby nedošlo k narušení jejich rozdělané práce.

```
UPDATE
    hr.person_extension
SET
    id_person = 158930 -- COO0001
WHERE
    id_person_extension IN (
          SELECT
```

```
            pe.id_person_extension
        FROM
            edu.study_relation AS sr
        JOIN hr.person_extension AS pe ON
            (sr.id_person_extension = pe.id_person_extension)
        JOIN hr.person AS p ON (pe.id_person = p.id_person)
        JOIN edu.study_programme AS sp ON
            (sp.id_study_programme = sr.id_study_programme)
        JOIN edu_study.study_state AS ss ON
            (ss.id_study_state = sr.id_current_study_state)
        WHERE
            p.login <> 'AUR0001'
            AND p.login <> 'MOR0030'
            AND p.login <> 'HAL191'
            AND p.login <> 'LIE78'
            AND p.login <> 'ABR01'
            AND p.login <> 'DOS78'
                -- FEI
            AND sr.id_faculty = 5
                -- bc., nav.
            AND (sp.id_study_type = 1 OR sp.id_study_type = 4)
                -- prezencni
            AND sr.id_study_form = 1
                -- Studuje v rocniku
            AND ss.id_study_state_definition = 3
);
```

Další ukázky SQL příkazů sloužících pro nastavování různých prerekvizit je možné nalézt v příloze na konci této práce a také na přiloženém DVD.

## 6.5 Tvorba testovacího scénáře v nástroji

Na základě identifikace části k testování a také na základě znalosti této oblasti se připraví testovací scénář. Zde jde hlavně o návrh chování uživatele a poté přesné sepsání jednotlivých kroků, které uživatel bude provádět. Před vytvářením samotného testu je také potřeba mít splněné všechny prerekvizity a systém připravený v takovém stavu, aby bylo možné test zaznamenat.

Samotný postup tvorby testu se odvíjí od zvoleného nástroje pro testování. Postupy pro vytváření testu jsou popsány u každého testovacího nástroje v třetí kapitole *Nástroje*

*pro výkonnostní a automatizované funkční testování*. Pro své testy jsem zvolil nástroje od IBM, konkrétně Rational Functional Tester a Rational Performance Tester, takže následující odstavce budou zaměřeny na ně.

### 6.5.1 Automatizované funkční testy v IBM Rational Functional Tester

U automatizovaných testů je kromě seznamu kroků vhodné mít také vymyšlené akce a části aplikace, které budou sloužit k ověřování správné funkčnosti. Při zaznamenávání testu se postupuje podle jednotlivých kroků testovacího scénáře. Pokud se dojde ke kroku, kde je třeba ověřit správnou funkčnost (typicky zobrazovaný výpis), tak se vytvoří verifikační bod. V případě, že bude třeba v kódu provést složitější operace, např. dynamicky procházet tabulku a provádět nějaké akce, je vhodné si zaznamenat daný prvek (např. tabulku), aby se s ním následně dalo pracovat v kódu. Nejlepším způsobem je vytvořit verifikační bod nad tímto prvkem, klidně takový, který nebude dávat relevantní výstup. Při úpravě scénáře v kódu potom tento verifikační bod smazat a tím pádem nepoužít, ale požadovaný objekt je již identifikován.

Po ukončení zaznamenávání testu, je vhodné ihned okomentovat vygenerovaný kód. Jedná se hlavně o komentáře u jednotlivých formulářových polí, protože při testování aplikace Edison, se tato pole identifikují podle vygenerovaného názvu ve stylu:

```
text_viewns_Z7_SHD09B1A0OBC40I()
```

Dále je vhodné z kódu eliminovat části, jako jsou např. přesné souřadnice při kliknutí na tlačítko, vyplňování formulářových polí pomocí stisků kláves apod. V případě, že náš test má využívat *Datapool*, nahradit vkládání informací přímo za data z úložiště *Datapool*.

Typickou akcí, která se u těchto testů vyskytuje, je dynamické procházení tabulky, a to ať už za účelem provádění nějaké akce, nebo za účelem postupného ověřování hodnot v tabulce. Data z tabulky můžeme v kódu získat následujícím způsobem:

```
ITestDataTable table = (ITestDataTable)
      table_viewns_Z7_SHD09B1A0OBC40().getTestData("contents");
for (int i = 0; i < table.getRowCount(); i++) {
      Object o = table.getCell(i, 1);
      TestDataText testDataText = (TestDataText)o;
      String text = data.getText();
}
```

### 6.5.2 Výkonnostní testy v IBM Rational Performance Tester

Při nahrávání testu se opět postupuje podle připravených kroků, podobně jako u funkčních testů. Mezi jednotlivými stránkami je třeba klikat pomalu, aby se jednotlivé požadavky k sobě správně začlenily. To usnadní práci a úpravy nahraného testu. Po skončení nahrávání aplikace sama provede tzv. korelaci dat a nabídne, jaké parametry se mají upravovat. Většinou jsou podchyceny parametry předávané pomocí metod GET a POST. Zde je ovšem důležitá znalost systému, jak bylo popsáno v jednom z předchozích bodů, protože je třeba zkontrolovat, zda se identifikovaly všechny potřebné parametry. V rámci informačního systému Edison jsou některé údaje přímo v URL a nikoliv jako

parametry HTTP protokolu. Takovéto údaje automatická korelace dat nepodchytí a je třeba je nahradit ručně.

Údaje nahrazujeme buď staticky, nebo dynamicky. Statické nahrazování se použije u přihlašovacích údajů, protože při testování systému Edison se používá typicky jenom jeden uživatel, z důvodů, které byly popsány dříve. Dynamické nahrazování je možné buď z vytvořeného úložiště *Datapool*, nebo výběrem pomocí regulárního výrazu z odpovědi některého z předchozích požadavků. U testů pro Edison je vhodné použít *Datapool* pro seznam studijních vztahů, který je při každém spuštění uživatele stejný. Nahrazování pomocí regulárního systému je poté nutné použít u údajů, které se mění pro každého uživatele (každý studijní vztah). Pro získání údaje pomocí regulárního výrazu je potřeba do testovacího scénáře vložit vlastní kód a předat mu v parametru odpověď odpovídajícího HTTP požadavku. Nalezení může probíhat následujícím kódem:

```
public String exec(ITestExecutionServices tes, String[] args) {
      String regExp ="\"concreteActivityId\":(.*?),";
      Pattern pattern = Pattern.compile(regExp);
      Matcher matcher  = pattern.matcher(args[0]);
      ArrayList<String> subjectIds = new ArrayList<String>();
      while(matcher.find()){
            subjectIds.add(matcher.group(1));
      }
      if(subjectIds.size()<1){
            if(tes.getTestLogManager()
                  .wouldReport(ITestLogManager.FILTER_WARNINGS)){
                  VerdictEvent ve = new VerdictEvent();
                  ve.setVerdict(VerdictEvent.VERDICT_ERROR);
                  ve.setReason(VerdictEvent.REASON_SEE_DESCRIPTION);
                  tes.getTestLogManager().reportEvent(ve);
            }
            tes.getLoopControl().continueLoop();
            return "-1";
      }
      return subjectIds.get(r.nextInt(subjectIds.size()));
}
```

Tento kód se dá použít prakticky pro nalezení jakéhokoliv údaje. Jediné, co se bude měnit, je regulární výraz. V systému Edison se lze setkat i se situací, že daný údaj nebude potřeba hledat v klasickém HTML kódu, který vrátil HTTP požadavek, ale např. v datovém formátu JSON. Takovýto případ nastává např. u volby rozvrhů, kde se data předávají pomocí AJAX ve formátu JSON, a to

proto, aby bylo docíleno co nejmenších datových nároků. Tento případ se ovšem neliší od vyhledávání v klasickém HTML kódu, pouze je třeba předat do našeho kódu v parametru správný obsah a odpovídajícím způsobem upravit regulární výraz.

## 6.6 Spouštění testu a monitorování systému

### 6.6.1 Automatizované funkční testy

Před samotným spouštěním funkčních testů, je potřeba test odladit a správně nastavit nástroj pro testování. Zde jde hlavně o vyladění časů, které bude mít aplikace vyhrazené mezi hledáním jednotlivých prvků v testované aplikaci, jako jsou např. tlačítka, formulářová pole a další. Správné nastavení je nutno odladit, aby test neselhal z důvodu, že testovací nástroj nenalezne nějaký prvek a tím pádem zhavaruje, i když testovaná část aplikace neobsahuje vady.

Co se týče monitorování systému během spuštěného testu, tak to u funkčních testů v podstatě není nutné, protože není třeba sledovat výkonnost a stabilitu systému a výsledek testu se vyhodnotí až z výsledného reportu.

### 6.6.2 Výkonnostní testy

Pro spouštění výkonnostních testů je potřeba doplnit ještě několik nastavení. Prvním nastavením jsou parametry testu, a to konkrétně kolik uživatelů a v jakém rozestupu se má spouštět. Pro odladění testu je vhodné začít s malým počtem uživatelů (např. tři uživatelé). Pro samotné testování je vhodné test spouštět několikrát a postupně zvyšovat zátěž tak, aby bylo možné identifikovat jednak přicházející problémy a jednak hranici stability systému. Při ladění testu se nebude sledovat zátěž na serveru, ale výsledný stav systému po skončení testu, tzn., jestli test proběhl v pořádku a provedly se všechny požadované akce. Pro ověření správného průběhu testu se může použít buď grafické rozhraní aplikace, kde se zkontroluje výsledný stav nebo v případě větších změn, pomocí dotazu do databáze. Například u testování přihlašování do rozvrhu se může u malého počtu uživatelů zkontrolovat výsledný stav po přihlášení do aplikace vizuální kontrolou zapsaných rozvrhových aktivit. Druhou možností je zjistit z databáze aktuální počet proběhlých voleb rozvrhů, spustit test a znovu provést dotaz nad databází a daná čísla porovnat. Ukázka SQL dotazu pro zjištění počet voleb rozvrhů:

```
SELECT
    MAX (id_concrete_activity_st_rel)
FROM
    common.concrete_activity_st_relation
;
```

Další věcí, kterou je před spuštěním testu nutné nakonfigurovat, jsou klienti, ze kterých se bude generovat zátěž. Pokud je test spouštěn z výkonného stroje, tak tento krok nemusí být nutný, ale pokud se při běhu testu narazí na hranici výkonu daného stroje, je třeba zátěž rozložit na více strojů.

Třetí věcí, kterou je vhodné si nastavit, je monitorování výkonu, které se bude provádět přímo z testovacího nástroje. Jednak je vhodné monitorovat výkon strojů, ze kterých se test spouští, aby bylo

možné odhalit případný problém s jejich výkonem a v tom případě navýšit jejich počet. Dále je výhodné přidat monitorování serverů a aplikací, které jsou testovány. Výhoda je v tom, že naměřené údaje jsou uložené v jednotlivých reportech testů a lze se k nim vrátit. U systému Edison je vhodné monitorovat z testovacího nástroje portálový server a aplikační server. Jako zdroj dat je v nástroji použit *IBM WebSphere PMI Monitoring* a konkrétně je vhodné nastavit tyto parametry:

- portal2-test.wps.vsb.cz
  - connectionPoolModule/wpdbJDBC_db2/FreePoolSize
  - connectionPoolModule/wpdbJDBC_db2/PoolSize
  - connectionPoolModule/wpdbJDBC_db2/WaitingThreadCount
  - jvmRuntimeModule/ProcessCpuUsage
  - jvmRuntimeModule/UsedMemory
  - threadPoolModule/WebContainer/ActiveCount
  - threadPoolModule/WebContainer/PoolSize
- as2-test.wps.vsb.cz
  - connectionPoolModule/DB2 EDISON/FreePoolSize
  - connectionPoolModule/DB2 EDISON/PoolSize
  - connectionPoolModule/DB2 EDISON/WaitingThreadCount
  - jvmRuntimeModule/ProcessCpuUsage
  - jvmRuntimeModule/UsedMemory
  - threadPoolModule/ORB.thread.pool/ActiveCount
  - threadPoolModule/ORB.thread.pool/PoolSize

Nutno podotknout, že přístup k těmto údajům je umožněný po přihlášení, a je proto nutné nastavit přístup daného uživatele (ideálně testovacího uživatele *coo0001*) do administračního rozhraní IBM WebSphere s rolí, která umožňuje monitorování.

Další možností monitorování systému během spuštěného testu je monitorování výkonu serverů pomocí linuxového nástroje *htop* a monitorování aplikačních parametrů pomocí administračního rozhraní IBM WebSphere.

## 6.7 Vyhodnocení výsledků

### 6.7.1 Automatizované funkční testy

U funkčních testů probíhá vyhodnocení jejich výsledků zpravidla na základě reportů, které test vygeneruje. Předpokladem pro správné vyhodnocení jsou správně zvolené verifikační body. V případě, že se v reportu objeví nějaká chyba nebo nějaký verifikační bod vrátí negativní výsledek, je potřeba identifikovat příčinu. V první řadě je nutné vyloučit možnost, že test selhal z důvodu špatného běhu nebo špatného návrhu, tzn., že příčinou není vada v testované aplikaci. Toto ověření může proběhnout pomocí ladění testu.

Pokud z reportu není přesně jasné, proč chyba nastala, můžou s identifikací problému pomoci aplikační logy. Pokud ani zde není žádná informace, která by vedla k závěru, proč test selhal, je nutné test upravit tak, aby kritickou oblast, ve které test selhal, pokryl více verifikačními body a pokud je to možné, tak upravit testovanou aplikaci pro detailnější logování.

Po odhalení příčiny selhání a následné úpravě aplikace, je nutné znovu spustit test z důvodu ověření, zda byla vada opravdu odstraněna a zda test proběhne bez chyb.

### 6.7.2 Výkonnostní testy

U výkonnostních testů je situace odlišná od funkčních testů. Vyhodnocování výsledků tady probíhá už v průběhu spuštěného testu, kdy je současně systém monitorován. První, co je třeba zkontrolovat je, zda test proběhl celý nebo se zastavil na některé stránce/akci v průběhu testování. Pokud k tomuto zaseknutí dojde, je velmi pravděpodobné, že portálový nebo aplikační server přestal odpovídat a je nedostupný. Nejjednodušším ověřením této domněnky je zkusit se přihlásit do systému přes webový prohlížeč.

Testovací nástroj dále zaznamenává během testu délku odpovědí na jednotlivé požadavky (stránky). Zde je velmi obtížné určit, jaké časy jsou ještě přijatelné a jaké časy již značí problém nebo jsou pro práci se systémem nepřijatelné. Očekávané časy se také odvíjí od počtu spouštěných uživatelů. Je nutné si uvědomit, že testovací prostředí obsahuje pouze jeden portálový server oproti produkčnímu prostředí, které obsahuje dva tyto servery. Z toho vyplývá, že dosažené časy s daným počtem uživatelů budou horší než při skutečném provozu na produkčním prostředí. Zde se nabízí myšlenka, že naměřené časy na testovacím prostředí budou o polovinu horší z důvodu chybějícího jednoho portálového serveru. Tato myšlenka, ale nemusí být správná, protože provoz může brzdit i aplikační server, který v testovacím prostředí odpovídá tomu v produkčním prostředí.

Pokud je detekován problém, je nutné odhalit příčinu. K tomu slouží sledování vytížení hardwarových prostředků serverů a vytížení aplikačních prostředků IBM WebSphere. Pokud dochází po delší dobu k maximálnímu vytížení procesorů nebo obsazení operační paměti na serverech, tak se nabízí navýšení hardwarových prostředků a znovu spuštění testů s cílem ověřit, zda je změna ve výkonu znatelná. Nutno ale podotknout, že v současné době jsou jednotlivé servery, na kterých běží systém Edison, hardwarově správně dimenzované a není potřeba prostředky navyšovat.

Jiná situace je u nastavení parametrů pro IBM WebSphere, kde je možnost velkého počtu různých nastavení. Z monitorování může být jasné na první pohled, v čem je problém, a který parametr je potřeba upravit nebo navýšit. Na druhou stranu správné vyladění systému tak, aby IBM WebSphere podávala co nejlepší výkony v každé situaci, je velká alchymie a úpravy probíhají spíše pokusnou metodou.

Samozřejmě problémem nemusí být jenom nastavení a hardwarové prostředky serveru, ale problém může být i v testované aplikaci. Identifikovat část aplikace (odpovídající stránku) může pomoci opět výstup z testovacího nástroje, a to podobným způsobem, jak bylo popsáno výše, tzn., že u daných požadavků je enormně velká doba odpovědi, případně běh testu na těchto požadavcích skončí. Pro identifikaci příčiny problému v testované aplikaci mohou, podobně jako u funkčních testů, opět pomoci i aplikační logy.

Poslední věcí, kterou je třeba hlídat během výkonnostního testování, je vytížení stroje, případně strojů, ze kterých se testování spouští. Pokud by tyto stroje dosáhly své hranice výkonnosti, tak to ovlivní průběh testů a výkonnostní výsledky testované aplikace nebudou odpovídat skutečnosti.

## 6.8 Shrnutí metodiky

Celou výše popisovanou metodiku lze shrnout do několika základních kroků, které je třeba při procesu testování projít. Jsou jimi:

- příprava testovacího prostředí – přepnutí aplikačních serverů (databáze),
- identifikace částí systému k testování – rozhodnutí o tom, co testovat,
- získání znalosti o dané části systému – zjištění funkčnosti systému,
- příprava prerekvizit – příprava databáze a dat,
- vytvoření testovacího scénáře – kroky testu a jejich záznam,
- nastavení parametrů testu a nástroje – úprava testu před spuštěním,
- opakované spouštění testu a monitorování systému – vlastní běh testu,
- vyhodnocení výsledků.

# 7 Ukázky automatizovaných funkčních testů

## 7.1 Test výpisu termínu zkoušky

### 7.1.1 Popis

Test ověřuje funkčnost výpisu termínu zkoušky. Ověřují se povinné položky formuláře a možnost výpisu termínu zkoušky se začátkem doby přihlašování až po zadaném termínu konání zkoušky. Následně proběhne ověření, zda se student na zkoušku zapíše.

### 7.1.2 Prerekvizity testu

Uživatel *rat0009* má v systému roli vyučujícího nebo garanta předmětu *Signály a soustavy (450-2009/01)*. Uživatel *coo0001* má v systému roli studenta tohoto předmětu, tzn., že jej má řádně zapsán ve svém osobním studijním plánu (stav předmětu je *Zapsán*). Uživatelé *rat0009* a *coo0001* musejí mít také platná hesla ve školním LDAP systému.

Test je nutné provádět v období 8. 11. 2014, 8:00 až 3. 12. 2014, 7:59. V případě provádění testu v jiném období, je potřeba upravit parametry vypsaných termínů odpovídajícím způsobem.

### 7.1.3 Akce testu

1. uživatel se přihlásí do systému Edison jako zaměstnanec pod uživatelem *rat0009*
2. uživatel se přes menu prokliká k možnosti výpisu zkoušek pro předmět *Signály a soustavy (450-2009/01)*
3. automaticky se odstraní všechny předem vypsané termíny zkoušek (z důvodu možnosti spouštění testu znovu bez manuálního zásahu)
4. uživatel přidá nový termín zkoušky s následujícím parametry:
   - datum začátku zkoušky: 4. 12. 2014
   - čas začátku zkoušky: 8:00
   - zaškrtnutí možnosti přihlašování studentů
   - datum začátku přihlašování: 8. 11. 2014
   - čas začátku přihlašování: 8:00
   - datum konce přihlašování: 4. 12. 2014
   - čas konce přihlašování: 8:00
   - kapacita: 0
   - areál: Poruba
   - budova: Fakulta elektrotechniky a informatiky
5. uživatel se pokusí uložit termín a ověří se, že se termín nepodaří uložit z důvodu chybějícího povinného parametru místnost
6. uživatel doplní parametr budova, vyplněný na hodnotu: EB429
7. uživatel uloží termín a ověří se, že se termín povedlo uložit
8. uživatel přidá druhý termín pomocí možnosti zkopírovat předchozí termín
9. uživatel nastaví datum začátku přihlašování na 8. 12. 2014
10. uživatel nastaví kapacitu na hodnotu 10
11. uživatel se pokusí uložit termín a dojde k ověření, že termín nelze uložit z důvodu, že termín začátku přihlašování na zkoušku je až po termínu konání zkoušky

12. uživatel nastaví datum začátku přihlašování na 3. 12. 2014
13. uživatel uloží termín a ověří se, že se termín povedlo uložit
14. uživatel přidá třetí termín pomocí možnosti zkopírovat předchozí termín
15. uživatel nastaví datum začátku přihlašování na 8. 11. 2014
16. uživatel uloží termín a ověří se, že se termín povedlo uložit
17. uživatel se odhlásí z SSO
18. uživatel se přihlásí do systému Edison jako student pod uživatelem *coo0001*
19. uživatel se přes menu dostane do nabídky přihlašování na zkoušky
20. ověří se, že jsou vypsané dané termíny
21. uživatel se pokusí přihlásit na třetí vypsaný termín (jediný, na který je mu to umožněno) a ověří se, že se uživatel úspěšně přihlásil
22. uživatel se odhlásí z SSO a zavře se okno prohlížeče

### 7.1.4 Očekávané výsledky

Všechny kroky testu projdou bez chyby. Nepodaří se vytvořit termín bez zvolené místnosti. Nepodaří se vytvořit termín, který má začátek přihlašování až po termínu konání zkoušky. Studentovi se zobrazí vypsané termíny a podaří se mu k třetímu vypsanému termínu přihlásit.

### 7.1.5 Dosažené výsledky

Dosažené výsledky odpovídaly očekávaným výsledkům, tzn., že test proběhl úspěšně a neodhalil žádnou vadu. Report z průběhu testu a samotný test jsou k nalezení na přiloženém DVD v *Ukázkové testy/IBM RFT/workspace/Projekt/FT_Zkouska.testsuite*.

## 7.2 Test zápisu bodů k úkolu

### 7.2.1 Popis

Test ověřuje funkčnost zápisu bodů za úkol studentům. Ověří se, že nelze zapsat více bodů než je limit a že se zapsané body uloží. Veškeré hodnoty testu (seznam studentů, jejich přidělované body a očekávaný výsledek) jsou uložené v úložišti *Datapool*. Test se automaticky opakuje tolikrát, kolik je údajů v úložišti *Datapool*.

### 7.2.2 Prerekvizity testu

Uživatel *rat0009* má v systému roli vyučujícího nebo garanta předmětu *Signály a soustavy (450-2009/01)*. Uživatelé uvedení v úložišti *Datapool* (zde konkrétně: *aug0012*, *auj0008* a *baj0057*) mají v systému roli studenta tohoto předmětu, tzn., že jej mají řádně zapsán ve svém osobním studijním plánu (stav předmětu je *Zapsán*). Uživatel *rat0009* musí mít také platné heslo ve školním LDAP systému.

### 7.2.3 Akce testu

1. uživatel se přihlásí do systému Edison jako zaměstnanec pod uživatelem *rat0009*
2. uživatel se přes menu prokliká k možnosti udělení bodů pro předmět *Signály a soustavy (450-2009/01)*
3. uživatel vyplní do pole *Os. číslo* hodnotu načtenou z úložiště *Datapool* podle klíče *login*

4. uživatel klikne na tlačítko *Vyhledat* a automaticky se doplní údaje studenta nalezeného podle osobního čísla
5. uživatel vyplní body podle hodnoty z úložiště *Datapool*, která odpovídá klíči *body*
6. podle hodnoty uložené v úložišti *Datapool* pod klíčem *result* se provede ověření chování systému
7. uživatel se odhlásí z SSO a zavře se okno prohlížeče

### 7.2.4 Očekávané výsledky

Všechny kroky testu projdou bez chyby. Nepodaří se zapsat výsledek s více body než je maximum. Pokud je počet bodů menší než maximum, tak se správně uloží a to platí i pro situaci, kdy počet bodů je nižší než minimální hranice pro úspěšné splnění daného úkolu.

### 7.2.5 Dosažené výsledky

Dosažené výsledky odpovídaly očekávaným výsledkům, tzn., že test proběhl úspěšně a neodhalil žádnou vadu. Report z průběhu testu a samotný test jsou k nalezení na přiloženém DVD v *Ukázkové testy/IBM RFT/workspace/Projekt/FT_Ukoly.testsuite*.

## 7.3 Test vyplnění kontaktní adresy

### 7.3.1 Popis

Test ověřuje funkčnost vyplnění kontaktní adresy studenta. Ověřují se povinné položky formuláře, zadání dat ve správném formátu, uložení údajů a zobrazení v historii údajů. Test také pracuje s daty v úložišti *Datapool*, ale je spuštěn pouze jednou a hodnoty v úložišti *Datapool* jsou iterovány v kódu.

### 7.3.2 Prerekvizity testu

Uživatel *coo0001* má v systému roli studenta a má platné heslo ve školním LDAP systému.

### 7.3.3 Akce testu

1. uživatel se přihlásí do systému Edison jako student pod uživatelem *coo0001*
2. uživatel se přes menu prokliká k možnosti úpravy kontaktních údajů
3. automaticky se smažou předchozí vyplněné hodnoty
4. začne se iterovat přes jednotlivé položky v úložišti *Datapool* a provedená akce se řídí podle hodnoty uložené v poli *result*
5. uživatel vyplní hodnoty z úložiště *Datapool* a podle položky *result* se ověří očekávaná funkčnost
6. po skončení vyplňování údajů z úložiště *Datapool*, uživatel klikne na možnost *Zpět s uložením*
7. dojde k ověření, že se zadané hodnoty uložily a zobrazují se v přehledu informací o uživateli
8. uživatel klikne na možnost *Upravit* u bloku kontaktních údajů
9. uživatel vyplní nové údaje a klikne na tlačítko *Uložit*
10. ověří se, že se podařilo údaje uložit

11. uživatel klikne na tlačítko *Zpět bez uložení*
12. provede se ověření, že v části *Historie adres* je uložena předchozí adresa
13. uživatel se odhlásí z SSO a zavře se okno prohlížeče

### 7.3.4 Očekávané výsledky

Všechny kroky testu projdou bez chyby. Formulář se nepodaří odeslat, pokud nejsou vyplněny všechny povinné položky nebo jsou vyplněny špatně (např. pro pole, kde se očekává číslo, jsou vyplněny jiné znaky). Správně vyplněné údaje se uloží a předešlá adresa se uloží do historie.

### 7.3.5 Dosažené výsledky

Dosažené výsledky odpovídaly očekávaným výsledkům, tzn., že test proběhl úspěšně a neodhalil žádnou vadu. Report z průběhu testu a samotný test jsou k nalezení na přiloženém DVD v *Ukázkové testy/IBM RFT/workspace/Projekt/FT_Adresa.testsuite*.

# 8 Ukázky výkonnostních testů

## 8.1 Testování přihlašování na zkoušky

### 8.1.1 Popis

Test vytváří zátěž na serveru formou zapisování na termíny zkoušek studenty. Používá se jeden student, který má přiřazeno několik studijních vztahů.

### 8.1.2 Prerekvizity testu

Uživatel *coo0001* má v systému přiřazené studijní poměry, které mají v aktuálním semestru zapsán předmět *Signály a soustavy (450-2009/01)*. Současně jsou pro tento předmět vypsány termíny zkoušek, na které je umožněno přihlašování (správné datum, volná místa). Uživatel *coo0001* musí mít také platné heslo ve školním LDAP systému.

### 8.1.3 Akce testu

1. uživatel se přihlásí do systému Edison jako student pod uživatelem *coo0001*
2. uživatel zvolí studijní vztah na základě ID, které je náhodně vybráno z připraveného úložiště *Datapool*
3. uživatel se prokliká na stránku s přihlašováním na zkoušky
4. uživatel se přihlásí na náhodně vybraný termín (náhodně se vybírají a nahrazují parametry *taskTermId* a *studentTaskId*)
    - může dojít i k odhlášení z termínu, pokud se náhodně vybraly parametry termínu, na kterém je uživatel již přihlášen
5. uživatel se odhlásí z SSO

### 8.1.4 Parametry testu

Test je spouštěn pro jednu skupinu uživatelů, kteří v nekonečné smyčce opakují kroky popsané výše. Test je vhodné spouštět na 3 až 5 minut. Počet uživatelů lze upravovat a zvyšovat, ale neměl by překročit počet přiřazených studijních vztahů pro uživatele *coo0001*. Uživatelé jsou také spouštěni postupně a prodleva mezi spouštěním jednotlivých uživatelů by se měla odvíjet od celkového počtu uživatelů.

### 8.1.5 Dosažené výsledky

Test proběhl bez chyby a naměřené odezvy, reporty, samotný test a další součásti jsou k nalezení na přiloženém DVD v *Ukázkové testy/IBM RPT/workspace/projekt/* pod názvem *PT_exams*.

## 8.2 Testování volby rozvrhů

### 8.2.1 Popis

Test vytváří zátěž na server při volbě rozvrhu studenty. Používá se jeden student, který má přiřazeno několik studijních vztahů. Uživatelé jsou rozděleni do tří skupin, které se postupně přihlásí, proklikávají stránky a ve stejném okamžiku začnou všichni volit rozvrh.

### 8.2.2 Předpoklady testu

Uživatel *coo0001* má v systému přiřazené studijní poměry, které mohou volit rozvrh. Jedná se o prezenční vztahy na fakultě FEI z bakalářského a magisterského studia. V systému je povolena volba rozvrhu pro všechny studijní vztahy. Uživatel *coo0001* musí mít také platné heslo ve školním LDAP systému.

### 8.2.3 Akce testu

1. uživatel se přihlásí do systému Edison jako student pod uživatelem *coo0001*
2. uživatel zvolí studijní vztah na základě ID, které je náhodně vybráno z připraveného úložiště *Datapool*
3. uživatel se prokliká na stránku rozvrhu a překlikává mezi možností *Volba rozvrhu* a *Volba sportu*
4. uživatel ve smyčce šestkrát zvolí předmět (náhodně se vybírá a nahrazuje parametr *selectStudyYearObligation*) a u daného předmětu ve smyčce volí dvě rozvrhové aktivity (náhodně se vybírá a nahrazuje parametr *selectConcreteActivity*)
    - může dojít i k odhlášení z rozvrhové aktivity, pokud se náhodně vybraly parametry tak, že na danou aktivitu je uživatel již přihlášen
5. uživatel se odhlásí z SSO

### 8.2.4 Parametry testu

Test je spouštěn pro tři skupiny uživatelů. Každá skupina má stejný počet uživatelů. Každé skupině je přiřazen test s výše popsanými kroky, mění se pouze časové prodlevy tak, aby bylo docíleno postupného připojování uživatelů a jednotného začátku volby rozvrhů. Test je nastaven tak, aby běžel do ukončení všech testovacích uživatelů. Počet uživatelů lze upravovat a zvyšovat, ale neměl by překročit počet přiřazených studijních vztahů pro uživatele *coo0001*. Uživatelé jsou spouštěni postupně a prodleva mezi spouštěním jednotlivých uživatelů by neměla být příliš velká, aby nedošlo k velkému časovému posunu.

### 8.2.5 Dosažené výsledky

Test proběhl bez chyby a naměřené odezvy, reporty, samotný test a další součásti jsou k nalezení na přiloženém DVD v *Ukázkové testy/IBM RPT/workspace/projekt/* pod názvem *PT_subjects*.

## 8.3 Testování průchodu systémem

### 8.3.1 Popis

Test vytváří zátěž na serveru při náhodném proklikávání jednotlivých stránek v systému ze studentského pohledu.

### 8.3.2 Předpoklady testu

Uživatel *coo0001* má v systému přiřazený alespoň jeden studijní poměr. Uživatel *coo0001* musí mít také platné heslo ve školním LDAP systému.

### 8.3.3 Akce testu

1. uživatel se přihlásí do systému Edison jako student pod uživatelem *coo0001*
2. uživatel ve smyčce desetkrát zobrazí stránku z horního menu první úrovně a na každé úrovni opět ve smyčce šestnáctkrát náhodně vybírá stránky, které si zobrazí
3. uživatel se odhlásí z SSO

### 8.3.4 Parametry testu

Test je spouštěn pouze pro jednu skupinu uživatelů, kteří v nekonečné smyčce provádí výše popsané kroky. Počet uživatelů lze upravovat a zvyšovat. Maximální hranice počtu uživatelů není omezena žádnými předpoklady testu. Test se může spouštět po libovolnou dobu. Doporučená doba je 5 až 10 minut. Náběh uživatelů může být jednotný anebo s nadefinovanou prodlevou. Je doporučeno zvolit menší časovou prodlevu, která se bude odvíjet od počtu zvolených uživatelů. Například pro sto uživatelů je vhodná prodleva 100 ms mezi začátky akcí jednotlivých uživatelů.

### 8.3.5 Dosažené výsledky

Test proběhl bez chyby a naměřené odezvy, reporty, samotný test a další součásti jsou k nalezení na přiloženém DVD v *Ukázkové testy/IBM RPT/workspace/projekt/* pod názvem *PT_pages*.

# 9 Průběh testování

## 9.1 Zadání

Z dlouhodobějšího monitorování systému bylo zjištěno, že při větší zátěži dochází k zaseknutí jednoho ze dvou portálových serverů po určitý čas. Během tohoto času jsou klienti obslouženi druhým portálovým serverem, ale dochází k degradaci výkonu, který se projeví koncovým uživatelům. Úkolem bylo navrhnout zátěžový test, pomocí kterého by bylo možné situaci nasimulovat a pokusit se zjistit v čem problém spočívá a následně jej vyřešit.

## 9.2 Přípravná fáze

Bylo rozhodnuto, že pro simulaci a vykonávání testu bude použito současné testovací prostředí systému Edison, které již existuje, ovšem jako databáze bude použita POKUSNA proto, aby nedošlo k velké změně databáze pro běžné testovací a analytické činnosti prováděné pracovníky oddělení CIT. Současně vzešel požadavek na správce jednotlivých serverů, aby povýšili počet procesorů a velikosti pamětí tak, aby hodnoty odpovídaly produkčnímu prostředí.

Pro simulaci situace byla vybrána akce volby rozvrhů, protože vytváří velkou zátěž a mnoho požadavků na systém a i v reálném provozu se u této akce projevily výkonnostní problémy, které se mají nasimulovat. Funkčnost zapisování rozvrhů jsem znal z uživatelského pohledu, ale bylo třeba tuto oblast prozkoumat z technického hlediska, zjistit jaké požadavky se volají, s jakými parametry apod. Bylo také třeba vypořádat se se situací, že je nutné spouštět několik uživatelů najednou, ale k dispozici je pouze jeden testovací uživatel. Tato situace byla vyřešena pomocí přidělení velkého počtu studijních vztahů k tomuto testovacímu uživateli.

Dalším krokem v přípravě bylo identifikovat prerekvizity pro bezchybné provedení testu. Bylo potřeba provést několik úprav v databázi. Konkrétně se jednalo o přiřazení správných studijních vztahů testovacímu uživateli, nastavení všech předmětů uživatele do stavu *Zapsán* a nastavení konce termínu volby rozvrhů tak, aby bylo možné měnit rozvrhové aktivity.

## 9.3 Tvorba testu

Pro tvorbu testu byl vybrán nástroj IBM Rational Performance Tester, který byl nainstalován na virtuální stroj *vsrvfeia0h-99.vsb.cz*, odkud byly testy spouštěny. Při návrhu testu, konkrétně jednotlivých kroků testu, nebyl kladen důraz na to, aby test simuloval přesné chování uživatelů při volbě rozvrhů, ale hlavně aby se podařilo vytvořit zátěž a zahltit systém požadavky. Byl tedy vytvořen test s následujícími akcemi:

1. uživatel se přihlásí do systému Edison jako student pod uživatelem *coo0001*
2. uživatel zvolí studijní vztah na základě ID, které je náhodně vybráno z připraveného úložiště *Datapool*
3. uživatel se prokliká na stránku rozvrhu, přejde na *Volba sportu* a poté zpět na *Volba rozvrhu*
4. uživatel ve smyčce šestkrát zvolí předmět (náhodně se vybírá a nahrazuje parametr *selectStudyYearObligation*) a u daného předmětu ve smyčce volí dvě rozvrhové aktivity (náhodně se vybírá a nahrazuje parametr *selectConcreteActivity*)

- může dojít i k odhlášení z rozvrhové aktivity, pokud se náhodně vybraly parametry tak, že na danou aktivitu je uživatel již přihlášen

5. uživatel přejde na stránku *Aktuální výsledky*
6. uživatel se odhlásí z SSO

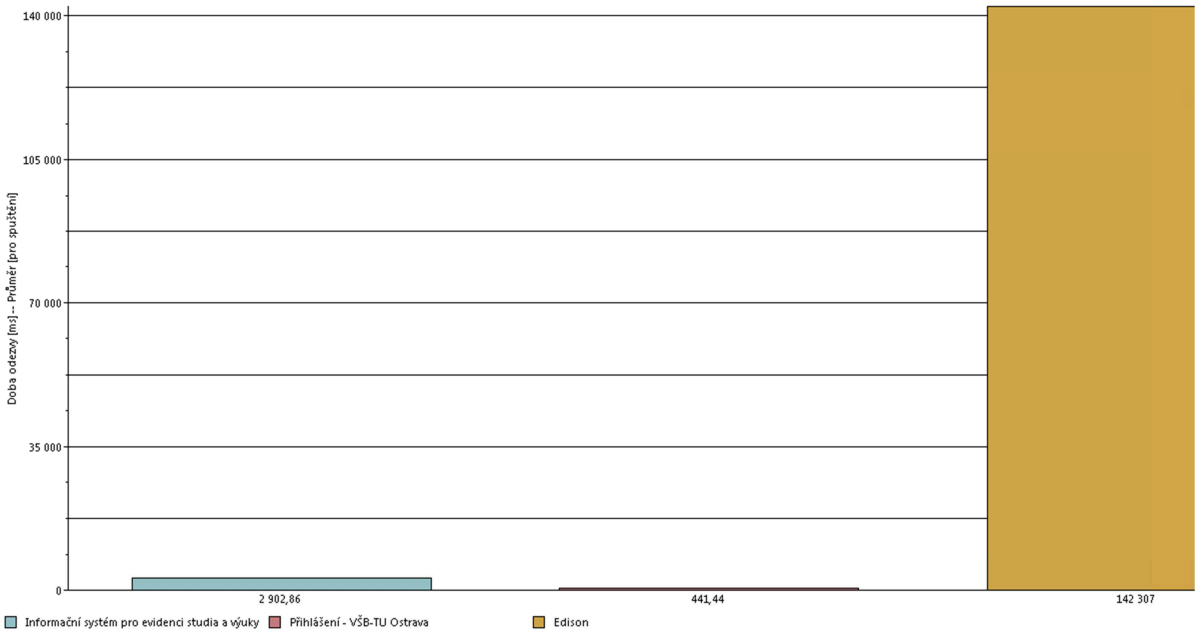
Test je poté spouštěn v nekonečné smyčce po dobu tří minut. Čas přemýšlení uživatele (*ThinkTime*) na jednotlivých stránkách je nastaven na jednu sekundu.

Po vytvoření testu byl test odladěn nad malým počtem uživatelů (3 uživatelé) a bylo ověřeno, že test probíhá správně a změny rozvrhů provedené testem se v databázi ukládají.

## 9.4 Testovací fáze

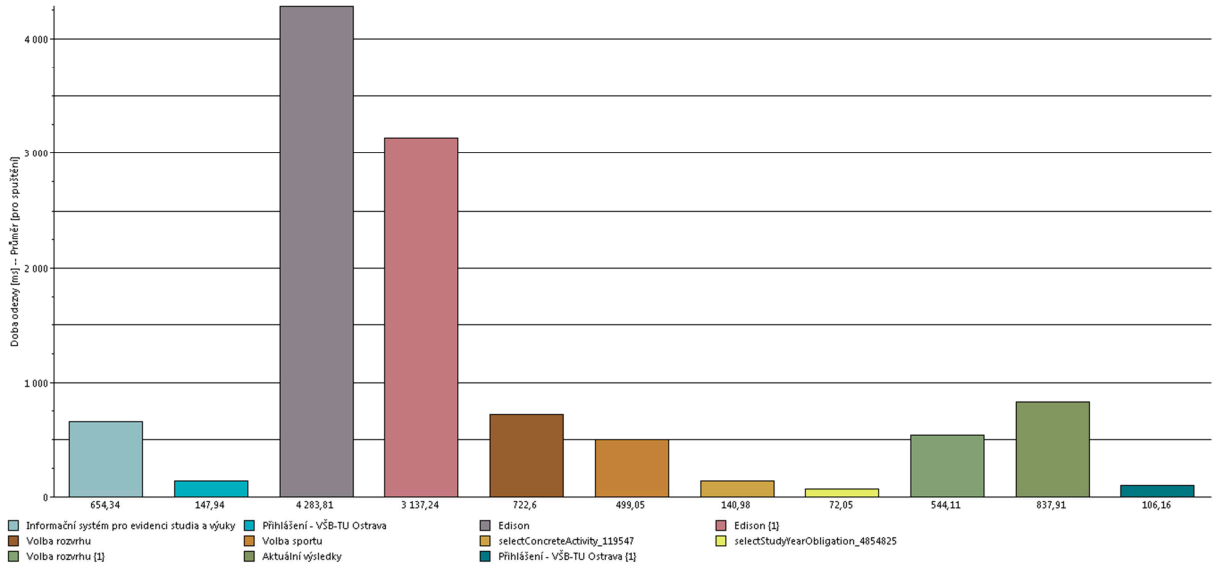
Během testování probíhalo monitorování zatížení virtuálního stroje, ze kterého se testovalo, pomocí standardního *Správce úloh systému Windows*. Dále bylo monitorováno vytížení portálového a aplikačního serveru pomocí linuxového nástroje *htop*. Samotná IBM WebSphere a její parametry byly monitorovány pomocí nástroje v administraci IBM WebSphere pro portálový i aplikační server. V testovacím nástroji byla sledována odezva na jednotlivé stránky a stav testu, jestli ještě pokračuje, nebo systém přestal odpovídat.

První test byl nastaven pro 50 uživatelů, kteří byli spouštěni najednou v jeden okamžik. Hned u tohoto testu došlo k zaseknutí portálového serveru, který přestal odpovídat. Tento stav byl zjištěn z grafů odpovědí jednotlivých stránek v testovacím nástroji (Obrázek 9.1), kde test skončil po akci přihlášení. Současně aplikace nebyla dostupná ani přes webové rozhraní.




Obrázek 9.1: *Odezvy stránek při testování 1*

Dalším úkolem tedy bylo zjistit hranici počtu uživatelů, při které dochází k zaseknutí a zjistit příčinu. Počet uživatelů byl snížen na 20 a test byl znovu spuštěn. Nyní vše proběhlo bez problémů, jak dokazuje graf na Obrázku 9.2.




Obrázek 9.2: *Odezvy stránek při testování 2*

Postupným zvyšováním počtu uživatelů bylo zjištěno, že hranice pro bezchybný průchod je 35 uživatelů. Z monitorování pomocí administračního rozhraní IBM WebSphere na portálovém serveru bylo zjištěno, že při zaseknutí dochází k vyčerpání poolů s volnými spojeními na portálové databáze. Tyto hodnoty tedy byly dvojnásobně navýšeny a test byl znovu spuštěn s kritickým počtem uživatelů. Z monitorování bylo zjištěno, že tentokrát test prošel. Následně testování pokračovalo se zvyšujícím se počtem uživatelů. Sérií několika testů byla odhalena další výkonnostní hranice na 75 uživatelích.

Při tomto počtu se narazilo na limit paměti na aplikačním serveru. Bylo zjištěno, že ač je na testovacím aplikačním serveru *as1-test.wps.vsb.cz* přidělena dostatečná paměť, tak nelze přidělit více paměti pro JVM, protože na serveru je nainstalován WAS s Express licencí, která je pouze 32 bitová. Pro další postup vznikl tedy nový testovací aplikační server *as2-test.wps.vsb.cz*, na kterém byla nainstalována verze WAS odpovídající produkčnímu aplikačnímu serveru. Současně zde byly také nastaveny stejné hardwarové prostředky.

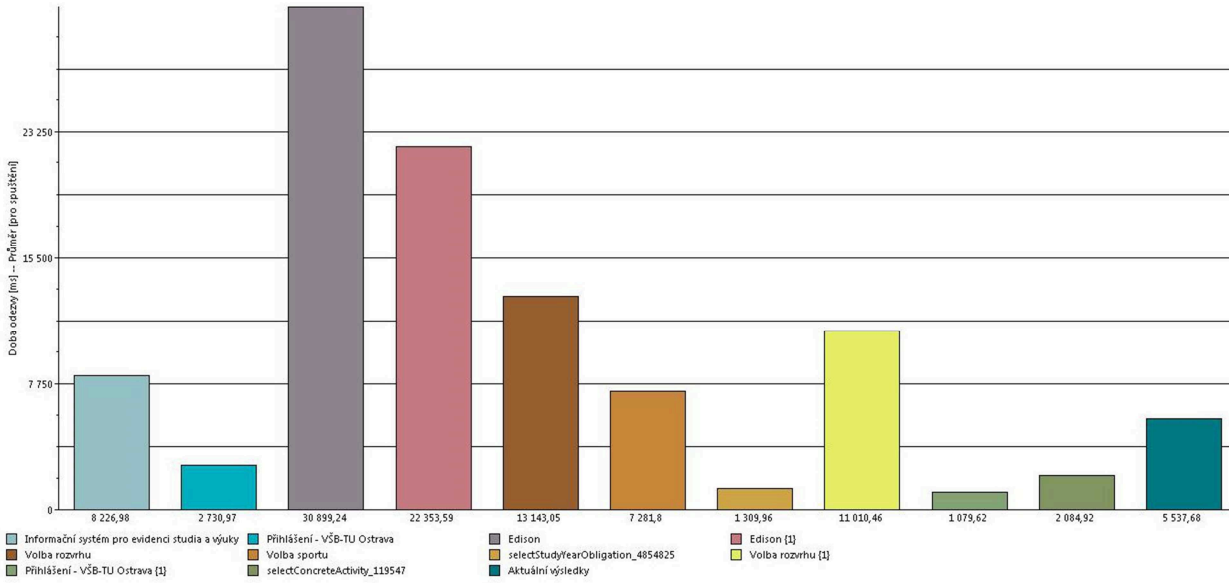
Při posledních testech s počtem uživatelů kolem 90, se přiblížil svému výkonnostnímu limitu také virtuální stroj, ze kterého byly testy spouštěny. Pro zajištění dalšího testování tak, aby výkon stroje neovlivňoval výsledek testování, bylo potřeba vybrat několik strojů, na které se nainstaluje agent testovacího nástroje, odkud se pak bude zátěž generovat. Agent byl nainstalován na devět počítačů na učebně EB213 v budově FEI. Současně byl agent také nainstalován na server *srvfeia0h.vsb.cz*, na kterém běží i virtuální stroj *vsrvfeia0h-99.vsb.cz*, odkud se testy spouštěly.

Po vytvoření, nainstalování a otestování druhého aplikačního serveru, testování pokračovalo. Před spuštěním testů bylo ovšem třeba přepnout testovací portálový server proti nově vzniklému aplikačnímu serveru *as2-test.wps.vsb.cz* a po skončení vyhrazeného času pro testování zase přepnout zpět proti původnímu aplikačnímu serveru *as1-test.wps.vsb.cz*.

Testy pokračovaly ověřením, zda navýšení pamětí pomohlo a zda nedojde k zaseknutí u 75 uživatelů. Test prošel a postupně se navyšoval počet uživatelů až k číslu 300, kde test neprošel celý a současně došlo k pádu SSO. Tento pád byl způsoben až nereálným zatížením, kdy se všech

300 uživatelů přihlašovalo v jeden jediný okamžik. Pro další testy byl změněn způsob nabíhání uživatelů tak, aby se nepřihlásili všichni najednou, ale s drobným rozestupem. Podle aktuálního nastavení počtu uživatelů se měnil parametr rozestupu mezi nabíháním uživatelů v rozmezí 50 - 100 ms.

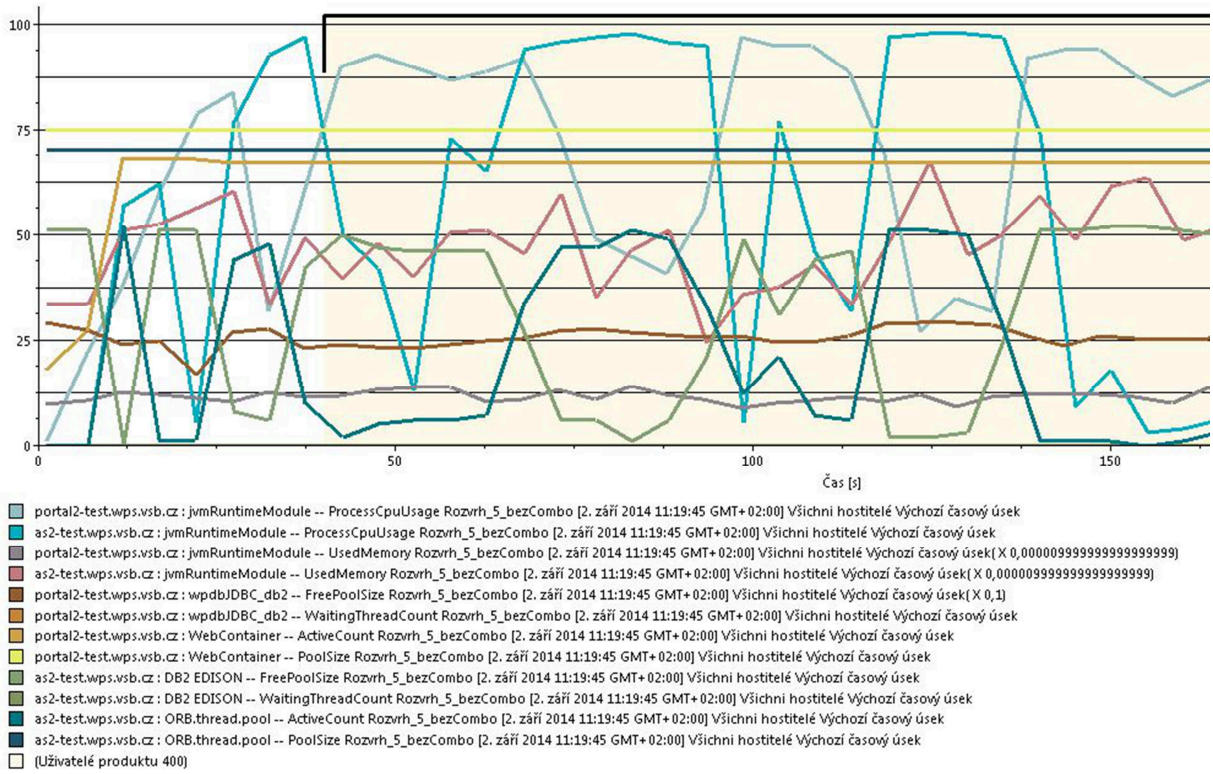
Postupným zvyšováním počtu uživatelů bylo dosaženo až 400 uživatelů, kteří si úspěšně navolili rozvrhy, a současně nedošlo k zaseknutí systému. Při tomto počtu uživatelů bylo dosaženo délek odezvy, které jsou na Obrázku 9.3.




Obrázek 9.3: *Odezvy stránek při testování 3*

Nejdelší odezvy byly na stránky po přihlášení a pro volbu studijního vztahu. Tyto stránky dosahovaly v průměru odezvy až 30 sekund. Ostatní stránky držely průměr okolo 10 sekund a samotná volba rozvrhů, tedy požadavky *selectStudyYearObligation* a *selectConcreteActivity*, dosahovaly odezvy v průměru okolo jedné, resp. dvou sekund.

Ukázka z monitoringu IBM WebSphere je na následujícím grafu na Obrázku 9.4 a jednotlivé hodnoty jsou v Tabulce 9.1.




Obrázek 9.4: *Monitorování zdrojů IBM WebSphere*

Tabulka 9.1: *Naměřené hodnoty IBM WebSphere*

| | portal2-test.wps.vsb.cz | as2-test.wps.vsb.cz |
|---|---|---|
| **jvmRuntimeModule/ProcessCpuUsage (průměr v %)** | 71,2 | 44,8 |
| **jvmRuntimeModule/ProcessCpuUsage (maximum v %)** | 97 | 98 |
| **jvmRuntimeModule/UsedMemory (průměr v kB)** | 1 161 707 | 4 880 892 |
| **jvmRuntimeModule/UsedMemory (maximum v kB)** | 1 420 880 | 6 729 148 |
| **WebContainer/ActiveCount (průměr)** | 64 | - |
| **WebContainer/ActiveCount (maximum)** | 68 | - |
| **WebContainer/PoolSize** | 75 | - |

| | | |
|---|---|---|
| **ORB.thread.pool/ActiveCount (průměr)** | - | 17,3 |
| **ORB.thread.pool/ActiveCount (maximum)** | - | 52 |
| **ORB.thread.pool/PoolSize** | - | 70 |
| **wpdbJDBC_db2/FreePoolSize (průměr)** | 260 | - |
| **wpdbJDBC_db2/PoolSize** | 292 | - |
| **DB2 EDISON/FreePoolSize (průměr)** | - | 35,4 |
| **DB2 EDISON/FreePoolSize (maximum)** | - | 52 |

Následně ještě byly provedeny pokusy s úpravou parametru *WebContaiter/PoolSize* na portálovém serveru za účelem zkrátit odezvu jednotlivých stránek. Na každou úpravu parametru navazovalo spuštění testu a ověření, zda se změna projevila na odezvě stránek. Původní hodnota poolu webového kontejneru byla 75. Při snaze hodnotu navýšit, bylo z dosažených výsledků zřejmé, že výkonnost je horší než při původním nastavení. Poté byla hodnota snížena a výsledky byly zhruba stejné jako při původním nastavení, a proto se nakonec nastavení tohoto parametru vrátilo do původního stavu.

Kompletní reporty, spouštěné testy a další součásti jsou uloženy na přiloženém DVD v *Praktické testování/workspace/EdisonTest/*.

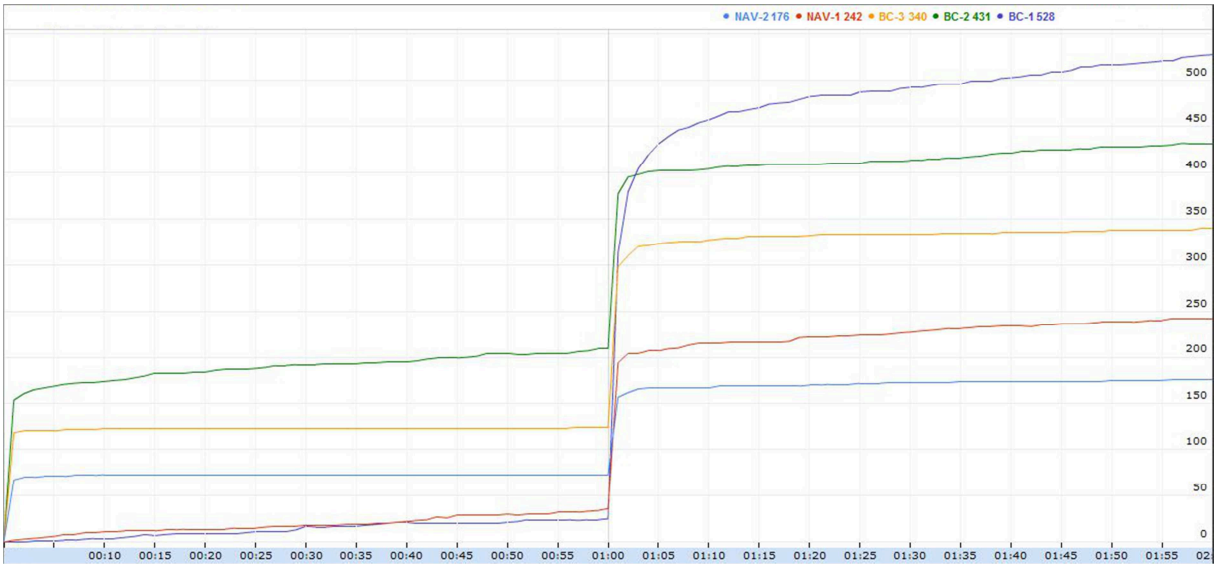
## 9.5 Zhodnocení testování

Po zhruba dvouměsíčním procesu testování se podařilo ustálit nastavení v takovém stavu, že bez zaseknutí serverů se 400 uživatelům podařilo zapisovat rozvrhy. Při reálném provozu zapisování rozvrhů, což je také součást systému Edison, při které je největší zatížení a nejvyšší počet souběžných uživatelů, bývá pro jeden ročník souběh zhruba 300 – 600 uživatelů s tím, že jsou rozděleni do dvou nestejně velkých skupin, přičemž zápis probíhá s hodinovým rozestupem. Z toho plyne, že nasimulovaná situace se 400 uživateli by měla být až za hranicí reálného vytížení, protože je třeba opět brát zřetel na to, že testy jsou spouštěny na testovacím prostředí, kde je pouze jeden portálový server.

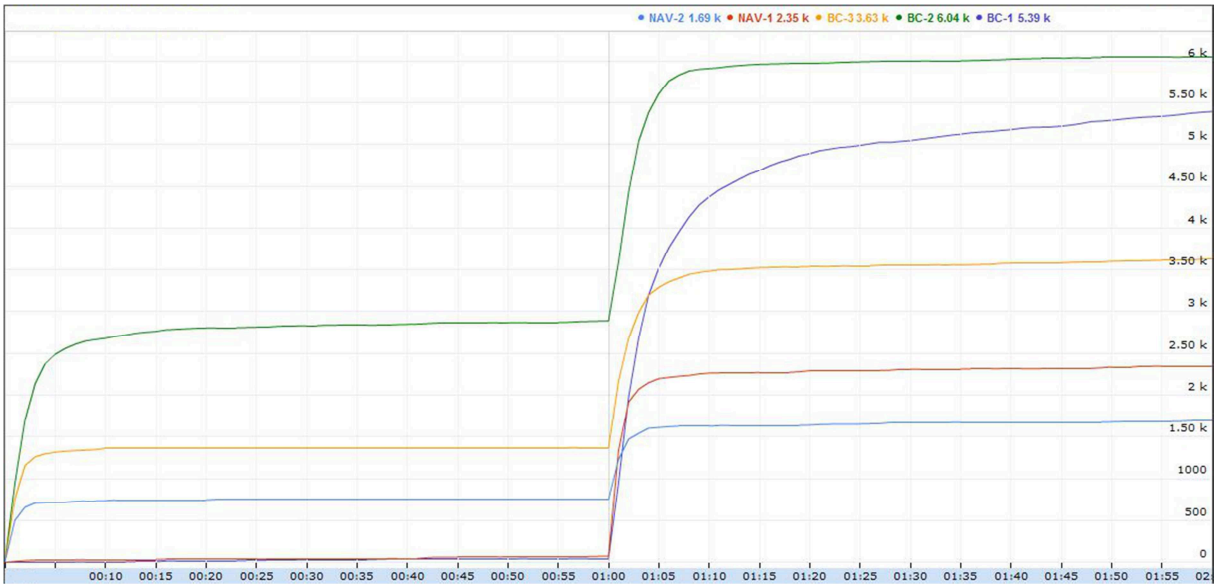
Výsledkem testování bylo, že se požadovanou kritickou situaci podařilo nasimulovat a následně se upravily parametry počtu spojení do portálových databází z portálového serveru. Jednalo se o úpravu parametru *Connection pool*, konkrétně hodnoty *Maximum connections*, která byla navýšena z původních 80 na 200 u portálových databází *community*, *customization*, *jcr* a *release*. Hodnota *Minimum connections* pro tyto databáze je 50. Dále u databází *feedback* a *likeminds*, které nejsou tak používané, zůstalo nastavení na původních hodnotách: *Minimum connections* je 10 a *Maximum connections* 50. Pravděpodobným důvodem proč docházelo k zaseknutí serveru je to,

že hodnota *Thread PoolSize* u webového kontejneru je nastavena na 75 a jedno vlákno si patrně bralo více spojení do databází, a proto původně nastavená hodnota 80 pro maximální počet spojení nestačila.

Po úpravách bylo znovu ověřeno pomocí testů, že ke kritické situaci již nedochází. V reálném provozu poté systém ustál volbu rozvrhů pro zimní semestr akademického roku 2014/2015 bez jakéhokoliv problému a většina studentů měla zapsaný rozvrh během prvních minut volby, jak ukazují následující grafy na Obrázku 9.5 a Obrázku 9.6.




Obrázek 9.5: *Počet studentů s alespoň jednou volbou*




Obrázek 9.6: *Počet voleb rozvrhových aktivit*

Při další volbě rozvrhů pro letní semestr akademického roku 2014/2015, byl systém samozřejmě také monitorován a dosažené výsledky byly velmi podobné, což opět potvrdilo, že systém je dostatečně stabilní a nápor uživatelů a požadavků bez potíží ustál.

# 10 Závěr

Cílem této diplomové práce bylo vytvořit metodické pokyny a ukázkové příklady, které by sloužily jako návod a referenční příklady pro budoucí vytváření testů nad platformou JavaEE s konkrétním zaměřením na informační systém Edison. Vytvořené postupy, pokyny a příklady vznikly jak pro automatizované funkční testování, tak pro výkonnostní testování.

Na začátku práce jsem uvedl několik základních pojmů týkajících se softwarového testování. Dále byly představeny a srovnány vybrané testovací nástroje. Pro každý nástroj jsem vytvořil popis a postup vytváření testů. Z vybraných nástrojů jsem vyhodnotil jako nejlepší IBM Rational Functional Tester pro automatizované funkční testy a IBM Rational Performance Tester pro výkonnostní testy. Tyto nástroje jsem také používal při tvorbě ukázkových testů a při praktickém testování.

Celá tato diplomová práce je zaměřena na informační systém Edison a je zde popsána infrastruktura produkčního i testovacího prostředí a jsou uvedeny možnosti monitorování a sběru dat, která jsou nezbytná pro vyhodnocování výkonnostních testů nad tímto systémem. Na základě obecných doporučení a procesů pro testování jsem vytvořil metodiku testování zaměřenou přímo na informační systém Edison, která popisuje jednotlivé kroky od přípravy testovacího prostředí až po vyhodnocení výsledků testování. K usnadnění tvorby testovacích scénářů a příkladů jsem vytvořil také několik ukázkových testů, na jejichž základě a principu lze vytvářet nové testy.

V rámci práce také proběhlo praktické testování systému Edison, při kterém byl řešen konkrétní výkonnostní problém. Během testování, které trvalo asi dva měsíce a probíhalo o letních prázdninách v roce 2014, bylo řešeno několik problémů, na které se během testování narazilo. Popis průběhu testování, řešení jednotlivých problémů, dosažené výsledky a následné úpravy systému jsou také popsány v této práci. Na přiloženém DVD jsou uloženy veškeré testy, plány testů a monitorovací výstupy z jednotlivých testů.

Snahou bylo, aby vytvořené metodické pokyny, postupy u testovacích nástrojů a další součásti této práce byly srozumitelné a aby s jejich pomocí byli vývojáři nebo jiní IT pracovníci, kteří zatím nemají s testováním zkušenosti, schopni testy pro systém Edison vytvářet a spouštět. Dle mého názoru se to podařilo a práce může sloužit k dalším účelům.

Dalším přínosem této práce bylo praktické výkonnostní testování, které proběhlo a které pomohlo odstranit výkonnostní problém, což bylo ověřeno i v produkčním prostředí při následných dvou zápisech rozvrhů studenty. Současně byly také identifikovány další oblasti, které by bylo vhodné výkonnostně otestovat a případně provést odpovídající změny. Tyto oblasti jsou Volba sportů studenty v systému Edison a systém jednotného přihlášení SSO. Tyto testy jsou naplánovány na letní prázdniny.

Tato práce měla pro mě osobní přínos v tom, že jsem poznal pozadí, infrastrukturu a provoz velkého systému, jakým je Edison. Současně jsem si také rozšířil své odborné znalosti o softwarové testování, což je podle mého názoru důležitá součást vývojového procesu.

# Použitá literatura


[1] BLACK, Rex. Advanced software testing – Vol 3.: Guide to the ISTQB Advanced Certification as an Advanced Technical Test Analyst. 1. vyd. Santa Barbara, CA: Rocky Nook, 2011, 586 s. ISBN 978-1-933952-39-0.

[2] FINK, Mark. The Hitchhiker's Guide to Test Automation: Software Quality Assurance and Test Automation in Practice. 1. vyd. CreateSpace Independent Publishing, 2013, 258 s. ISBN 978-1456486129.

[3] BEIZER, Boris. Black-box Testing: Techniques for functional testing of software and systems. 1. vyd. New York: John Wiley & Sons, Inc., 1995, 294 s. ISBN 0-471-12094-4.

[4] BOURQUE, Pierre a FAIRLEY, Richard E. Swebok: Guide to the Software Engineering Body of Knowledge. Version 3.0. Los Alamitos, CA: IEEE Computer Society, 2014, 294 s. ISBN 07-695-5166-1.

[5] Apache JMeter™. [online]. [cit. 2015-03-02]. Dostupné z: http://jmeter.apache.org/

[6] Apache JMeter. [online]. [cit. 2015-03-02]. Dostupné z: http://en.wikipedia.org/wiki/Apache_JMeter

[7] Eclipse Test & Performance Tools Platform Project. [online]. [cit. 2015-03-04]. Dostupné z: https://eclipse.org/tptp/

[8] SeleniumHQ Browser Automation. [online]. [cit. 2015-03-05]. Dostupné z: http://www.seleniumhq.org/

[9] Selenium History. [online]. [cit. 2015-03-05]. Dostupné z: http://www.seleniumhq.org/about/history.jsp

[10] Rational Functional Tester. [online]. [cit. 2015-03-07]. Dostupné z: http://www-03.ibm.com/software/products/cs/functional

[11] Rational Functional Tester. [online]. [cit. 2015-03-07]. Dostupné z: http://en.wikipedia.org/wiki/Rational_Functional_Tester

[12] Přehled produktu Rational Performance Tester. In: IBM Knowledge Center [online]. [cit. 2015-03-09]. Dostupné z: http://www-01.ibm.com/support/knowledgecenter/SSMMM5_8.7.0/com.ibm.rational.test.lt.doc/topics/c_rpt_ovr.html

[13] Rational Performance Tester. [online]. [cit. 2015-03-09]. Dostupné z: http://en.wikipedia.org/wiki/Rational_Performance_Tester

[14] Exploratory Testing Explained. [online]. [cit. 2015-03-15]. Dostupné z: http://www.satisfice.com/articles/et-article.pdf

[15] Základy automatizace testování. [online]. [cit. 2015-03-15]. Dostupné z: http://www.swtestovani.cz/index.php/uvod-do-testovani/40-zaklady-automatizace-testovani

[16] Functional Testing. [online]. [cit. 2015-03-15]. Dostupné z: http://www.techopedia.com/definition/19509/functional-testing

[17] Funkční a nefunkční testy. [online]. [cit. 2015-03-15]. Dostupné z: http://testovanisoftwaru.cz/druhy-typy-a-kategorie-testu/funkcni-a-nefunkcni-testy/

[18] Nagios. [online]. [cit. 2015-04-17]. Dostupné z: http://en.wikipedia.org/wiki/Nagios

[19] Operating principle of Nagios. [online]. [cit. 2015-04-17]. Dostupné z: http://en.wikipedia.org/wiki/Nagios#/media/File:Monitoring.png

[20] Cacti – The complete rrdtool-based Graphing Solution. [online]. [cit. 2015-04-17]. Dostupné z: http://www.cacti.net/what_is_cacti.php

[21] Cacti Working Principle. [online]. [cit. 2015-04-17]. Dostupné z: https://xuri.me/wp-content/uploads/2013/10/install-the-cacti-server-monitor-on-ubuntu-server-17.jpg

[22] Hyperic HQ Wiki. [online]. [cit. 2015-04-19]. Dostupné z: http://hyperic-hq.software.informer.com/wiki/

[23] Hyperic Overview. [online]. [cit. 2015-04-19]. Dostupné z: https://support.hyperic.com/display/EVO/Hyperic+Overview

[24] Hyperic Architecture Overview. [online]. [cit. 2015-04-19]. Dostupné z: https://support.hyperic.com/download/attachments/71828418/archoverview.gif

# Seznam příloh



Součástí DP je DVD.

Obsah přiloženého DVD:

1. Diplomová práce.docx - *vlastní práce*
2. Diplomová práce.pdf - *vlastní práce*
3. skripty.sql – *SQL skripty pro nastavení prerekvizit*
4. Praktické testování/studyRelations.csv – *CSV soubor se studijními vztahy uživatele coo0001*
5. Praktické testování/workspace/EdisonTest/ – *veškeré testy, výsledky a další součásti použité při praktickém testování*
6. Ukázkové testy/IBM RFT/workspace/Projekt/ – *ukázkové automatizované funkční testy*
7. Ukázkové testy/IBM RPT/workspace/projekt/ – *ukázkové výkonnostní testy*
8. Ukázkové testy/Ostatní nástroje/ – *testy vytvořené v nástrojích Eclipse TPTP, JMeter a Selenium IDE*

# Příloha.A: Ukázka nastavení prerekvizit – výběr studijních vztahů

SQL příkaz pro výběr studijních vztahů. Vyberou se jenom ty vztahy, které mají zapsán alespoň jeden předmět. Typicky lze výsledek tohoto příkazu exportovat do CSV souboru a naplnit jím *Datapool* pro daný test.

```
SELECT
    pe.id_person_extension AS id_study_relation
FROM
    hr.person_extension AS pe
    JOIN edu.study_relation AS sr ON
      (sr.id_person_extension = pe.id_person_extension)
    JOIN edu.study_programme AS sp ON
      (sr.id_study_programme = sp.id_study_programme)
    JOIN edu.study_type AS st ON
      (st.id_study_type = sp.id_study_type)
    JOIN edu_study.study_class_year AS scy ON
      (scy.id_study_class_year = sr.id_last_opened_stu_clas_year)
    JOIN edu.class_year AS cy ON
     (cy.id_class_year = scy.id_class_year)
WHERE
    pe.id_person = 158930 -- COO0001
    AND EXISTS (
        SELECT
           *
        FROM
            edu_study.study_year AS sy
            JOIN edu_study.study_year_obligation AS syo ON
                (syo.id_study_year = sy.id_study_year)
            JOIN edu_study.subject_block_assignment AS sba ON
                (sba.id_subject_block_assignment = syo.id_subject)
        WHERE
            sy.id_study_relation = sr.id_person_extension
            AND sy.id_academic_year = 53 -- 2013/2014
```

```
            AND syo.id_semester = 111 -- 2013/2014 - leto
            AND EXISTS (
                 SELECT
                    *
                 FROM
                    common.concrete_activity AS ca
                 JOIN edu_study.subject_version_study_form AS svsf
                     ON (svsf.id_subject_version_study_form =
                           ca.id_sub_version_study_form)
                 WHERE
                     ca.capacity > 0
                     AND ca.id_semester = 111 -- 2013/2014 - leto
                     AND svsf.id_subject_version =
                           sba.id_subject_version
            )
        )
ORDER BY
        pe.id_person_extension
;
```

# Příloha.B: Ukázka nastavení prerekvizit – nastavení stavu předmětů

SQL příkaz, který nastaví všem předmětům u studentů z fakulty FEI, prezenčního bakalářského a magisterského studia v akademickém roce 2013/2014 stav předmětů na *Zapsán*. Úprava se nedotkne vyjmenovaných uživatelů, kterými jsou zaměstnanci oddělení CIT, aby nedošlo k narušení jejich rozdělané práce.

```
UPDATE
    edu_study.stu_year_obl_state_ass
SET
    id_stu_year_obl_state = 7 -- Zapsana
WHERE
    id_study_year_obligation IN (
         SELECT
```

```
        syo.id_study_year_obligation
    FROM
        edu_study.study_year_obligation AS syo
        JOIN edu_study.study_year AS sy ON
            (syo.id_study_year = sy.id_study_year)
        JOIN edu.study_relation AS sr ON
            (sy.id_study_relation = sr.id_person_extension)
        JOIN hr.person_extension AS pe ON
            (sr.id_person_extension = pe.id_person_extension)
        JOIN hr.person AS p ON (pe.id_person = p.id_person)
        JOIN edu.study_programme AS sp ON
            (sp.id_study_programme = sr.id_study_programme)
        JOIN edu_study.study_state AS ss ON
            (ss.id_study_state = sr.id_current_study_state)
    WHERE
        AND p.login <> 'AUR0001'
        AND p.login <> 'MOR0030'
        AND p.login <> 'HAL191'
        AND p.login <> 'LIE78'
        AND p.login <> 'ABR01'
        AND p.login <> 'DOS78'
            -- FEI
        AND sr.id_faculty = 5
            -- bc., nav.
        AND (sp.id_study_type = 1 OR sp.id_study_type = 4)
            -- prezencni
        AND sr.id_study_form = 1
            -- Studuje v rocniku
        AND ss.id_study_state_definition = 3
            -- 2013/2014
        AND sy.id_academic_year = 53
    )
;
```

# Příloha.C: Ukázka nastavení prerekvizit – promazání navolených rozvrhů

SQL příkaz promaže u všech studijních vztahů uživatele *coo0001* všechny zvolené rozvrhové aktivity.

```
DELETE FROM
    common.concrete_activity_st_relation
WHERE
    id_concrete_activity_st_rel IN (
        SELECT
            casr.id_concrete_activity_st_rel
        FROM
            common.concrete_activity_st_relation AS casr
            JOIN hr.person_extension AS pe ON
               (casr.id_study_relation = pe.id_person_extension)
            JOIN common.concrete_activity AS ca ON
               (ca.id_concrete_activity = casr.id_concrete_activity)
            JOIN edu.semester AS sem ON
               (sem.id_semester = ca.id_semester)
        WHERE
            pe.id_person = 158930 -- COO0001
            AND sem.id_academic_year = 53 -- 2013/2014
    )
;
```

# Příloha.D: Ukázka nastavení prerekvizit – nastavení termínu volby rozvrhů

SQL příkaz nastaví konec volby rozvrhů pro letní semestr 2013/2014 na datum 31. 8. 2014.

```
UPDATE
    edu_study.schedule_sel_term_conf
SET
    end_timestamp = '2014-08-31 23:59:59.0'
WHERE
    id_schedule_sel_term_conf = 111 -- 2013/2014 - leto
;
```